NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN OF A DIGITAL INTERFEROMETRIC DEMODULATOR

By

David William Brenner

September, 1994

Thesis Advisor: Robert M. Keolian

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE September 1994	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE DESIGN OF A DIGITAL INTERFEROMETRIC DEMODULATOR		5. FUNDING NUMBERS
6. AUTHOR(S) Brenner, David, William		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES. The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE A

13. ABSTRACT (maximum 200 words)

This thesis investigates the design of a digital fiber-optic interferometric demodulator based upon a passive symmetric combination of signals produced by an interferometer terminated with a 3x3 optical coupler. The demodulator was implemented using a digital signal processing (DSP) board based upon a TMS320C31 DSP processor. The demodulator was tested using signals produced by a set of intereferometric simulators which used an analog AD639 trigonometric chip. A goal of this thesis was to demonstrate an improved noise floor at low frequencies as compared with a similar analog implementation.

14. SUBJECT TERMS Demodulation, Fiber Optic, Interferometry, Hydrophone			15. NUMBER OF PAGES 106
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18 298-102

DESIGN OF A DIGITAL INTERFEROMETRIC DEMODULATOR

by

David W. Brenner
Lieutenant Commander, Canadian Forces
B.Eng., Royal Military College of Canada, 1983

Submitted in partial fulfillment
of the requirements for the degrees of

MASTER OF SCIENCE IN ENGINEERING ACOUSTICS

from the

NAVAL POSTGRADUATE SCHOOL
September, 1994

Author: [illegible]
David W. Brenner

Approved by: [illegible]
Robert M. Keolian, Thesis Advisor

[illegible]
Steven L. Garrett, Second Reader

[illegible]
James V. Sanders, Acting Chairman,
Engineering Acoustics Academic Committee

ABSTRACT

This thesis investigates the design of a digital fiber-optic interferometric demodulator based upon a passive symmetric combination of signals produced by an interferometer terminated with a 3x3 optical coupler. The demodulator was implemented using a digital signal processing (DSP) board based upon a TMS320C31 DSP processor. The demodulator was tested using signals produced by a set of intereferometric simulators which used an analog AD639 trigonometric chip. A goal of this thesis was to demonstrate an improved noise floor at low frequencies as compared with a similar analog implementation.

TABLE OF CONTENTS

I. INTRODUCTION 1
A. MOTIVATION 1
B. GOAL 2
C. BACKGROUND 2
1. Interferometric Sensors 2
2. Symmetric Demodulation 6
II. DESIGN OF A SYMMETRIC DIGITAL DEMODULATOR 9
A. INITIAL DEMODULATOR DESIGN 9
1. Signal Digitization. 9
2. Algorithm Considerations. 10
B. SIMULATION ON AN IBM PC USING TURBO C 11
C. SIMULATION USING MATLAB 18
D. PROGRAMMING THE PC31 22
1. TMS320 Optimizing C Compiler Programs 22
2. TMS320 Assembler Programs 24
III. PERFORMANCE OF THE DIGITAL DEMODULATOR 27
A. TESTING OF THE INPUT SAMPLING CIRCUITS. 27
1. Data for Full Scale Inputs 27
2. Data for Inputs Grounded. 28
B. SAMPLING OF THE INTERFEROMETRIC SIGNAL SIMULATORS 33
C. DIGITAL DEMODULATOR OUTPUT. 36
1. Demodulator Response to Input Frequency Variations. 36
2. Demodulator Response to Optical Phase Shift Variations. 41
IV. DESIGN OF AN EXTERNAL SAMPLE AND HOLD CIRCUIT 49
A. DESIGN 49
B. RESULTS OF CIRCUIT TESTING 51
V. CONCLUSIONS AND RECOMMENDATIONS. 55
APPENDIX A. DSP CARD: PC31 BY INNOVATION INTEGRATION 57

APPENDIX B. ANALOG INTERFEROMETRIC SIGNAL SIMULATOR 59
APPENDIX C. MISCELLANEOUS 61
A. PROGRAM TCDEMOD.C 62
B. PROGAM PLOTDAT.M 65
C. PROGRAM DEMOD.M 66
D. PROGRAM PLOTABC.M 69
E. PROGRAM LDTST3.M 71
F. PROGRAM LDTST7.M 72
G. PROGRAM TMSDEMOD.C 74
H. PROGRAM TST3.ASM 76
I. PROGRAM TST5.ASM 80
J. PROGRAM TST7.ASM 87
REFERENCES. 95
INITIAL DISTRIBUTION LIST 97

I. INTRODUCTION

A. MOTIVATION

The motivation for this work is in support of the fiber optic hydrophone research being conducted at the Naval Postgraduate School. Push-pull fiber-optic hydrophones that have been designed and built by Garrett, Brown and associates at the Naval Postgraduate School offer very large acoustic sensitivities [Refs 1, 2, 3, 4 and 5]. This has offered the potential to develop acoustic sensor systems with wide dynamic ranges and high sensitivity. Various demodulation schemes are feasible for use with these interferometric sensors [Ref 6]. In particular, Cameron [Ref 7], has demonstrated an approach which terminates a Mach-Zehnder interferometer with a 3x3 optical coupler as shown in Figure 1.1. By using all three optical outputs and a passive symmetric demodulation scheme developed at the Naval Postgraduate School, he demonstrated superior results over other passive homodyne implementations. The algorithm is shown in block diagram form in Figure 1.2, below. Cameron constructed an interferometric demodulator based on the passive symmetric algorithm using analog electronics. With such, he was able to achieve a dynamic range of 115 dB @ 600 Hz in a one Hertz bandwidth with no more than 4% Total Harmonic Distortion (THD) [Ref 7:p 252]. The minimum detectable signal at that same frequency was 220 μradians/√Hz. Also Brown, et al., [Ref 8] quote comparable performance using a similar analog symmetric demodulator. They also note that both the minimum detectable signal (noise floor) and maximum tolerable signal increase as the acoustic frequency decreases.

At low frequencies, the analog electronic version of the symmetric demodulator suffered from excessive noise. When comparing the relative importance of noise contributions from various stages of a circuit, it is advantageous to refer the noise to a common spot in the circuit, for instance the input. This is done by dividing the noise injected at a stage by the accumulated gain of all the previous stages. The excessive low frequency noise was a consequence of the analog multiplier stage which follows the differentiators (refer ahead to Figure 1.2). A differentiator has gain which is proportional to frequency, going to zero at dc. Therefore, any noise in the multipliers at low frequencies, will be of great importance; the noise power will diverge at dc as one over the frequency squared. The analog multipliers available for use in the analog symmetric demodulator were not particularly quiet. They had noise levels which were roughly equivalent to 16 bit noise in digital systems. However, the problem should be dramatically reduced if the symmetric demodulation can implemented on a high speed Digital Signal Processor (DSP), where multiplication can be performed with 32 bit or better precision. For this reason it is hoped that a digital implementation will perform better than the analog version, and not have the dynamic range at low frequencies limited by excessive noise. McGinnis [Ref 9] investigated the use of digital algorithms to implement the symmetric demodulation scheme. He successfully programmed the algorithm in the LabVIEW^{TM} programming language, and was able to demonstrate increased performance over the analog implementation (133 dB dynamic range @ 1 kHz 4% THD using a 51.2 kHz sampling rate).

B. GOAL

The goal of the thesis was to design, implement and test a digital fiber-optic interferometric demodulator using the passive symmetric demodulation scheme. Ultimately, a stand alone digital demodulator was to be built for use in the field. The performance of the demodulator was to be tested and compared against the analog version of the symmetric demodulator and the computer simulation conducted by McGinnis.

C. BACKGROUND

1. Interferometric Sensors

Interferometric sensors developed at the Naval Postgraduate School for sensing acoustic fields have been mostly based on the Mach-Zehnder two-beam interferometer, as illustrated in Figure 1.1. The sensor is comprised of a low power semiconductor laser which illuminates one leg of a 2x2 optical coupler. The output of the optical 2x2 coupler comprises the two legs of the Mach-Zehnder interferometer. Each leg is strained differentially by the field to be measured so that the two legs operate in a push-pull manner. The optical path length changes in the two legs cause interference in the 3x3 optical coupler whose output can be fed to photo diodes for detection and demodulation.

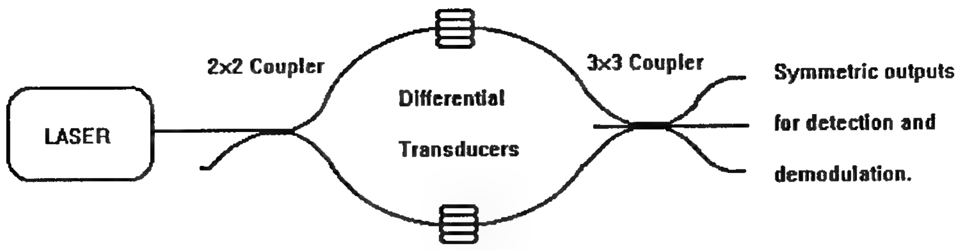

Figure 1.1. Schematic Diagram of a Fiber Optic Mach-Zehnder Interferometric Sensor

An ideal 3x3-coupler provides three signals of equal amplitude, each with a relative phase difference of 120°. The light intensities of each output leg from the 3x3-coupler are:

$$a = A + B\cos\left[\phi(t)\right], \tag{1.1}$$

$$b = A + B\cos\left[\phi(t) + 120°\right], \tag{1.2}$$

$$c = A + B\cos\left[\phi(t) + 240°\right], \tag{1.3}$$

where A is the average value of the optical intensity emitted by the optical fibers and B is the peak deviation in optical intensity from the average value A. The signal of interest, ϕ (t) is assumed to be time harmonic:

$$\phi(t) = \sin(\omega t) + \phi_{drift}(t). \tag{1.4}$$

The $\phi_{drift}(t)$ term accounts for the phase fluctuations due to environmental effects (hydrostatic pressure fluctuations and temperature changes) plus the static phase shift due to the physical differences in the two optical path lengths. These fluctuations are assumed to be slowly varying in comparison to the signal of interest and are therefore normally ignored. A theoretical plot of a 3x3 coupler output with constant A and B is shown in Figure 1.3. The three wave forms have been displaced from one another for clarity.

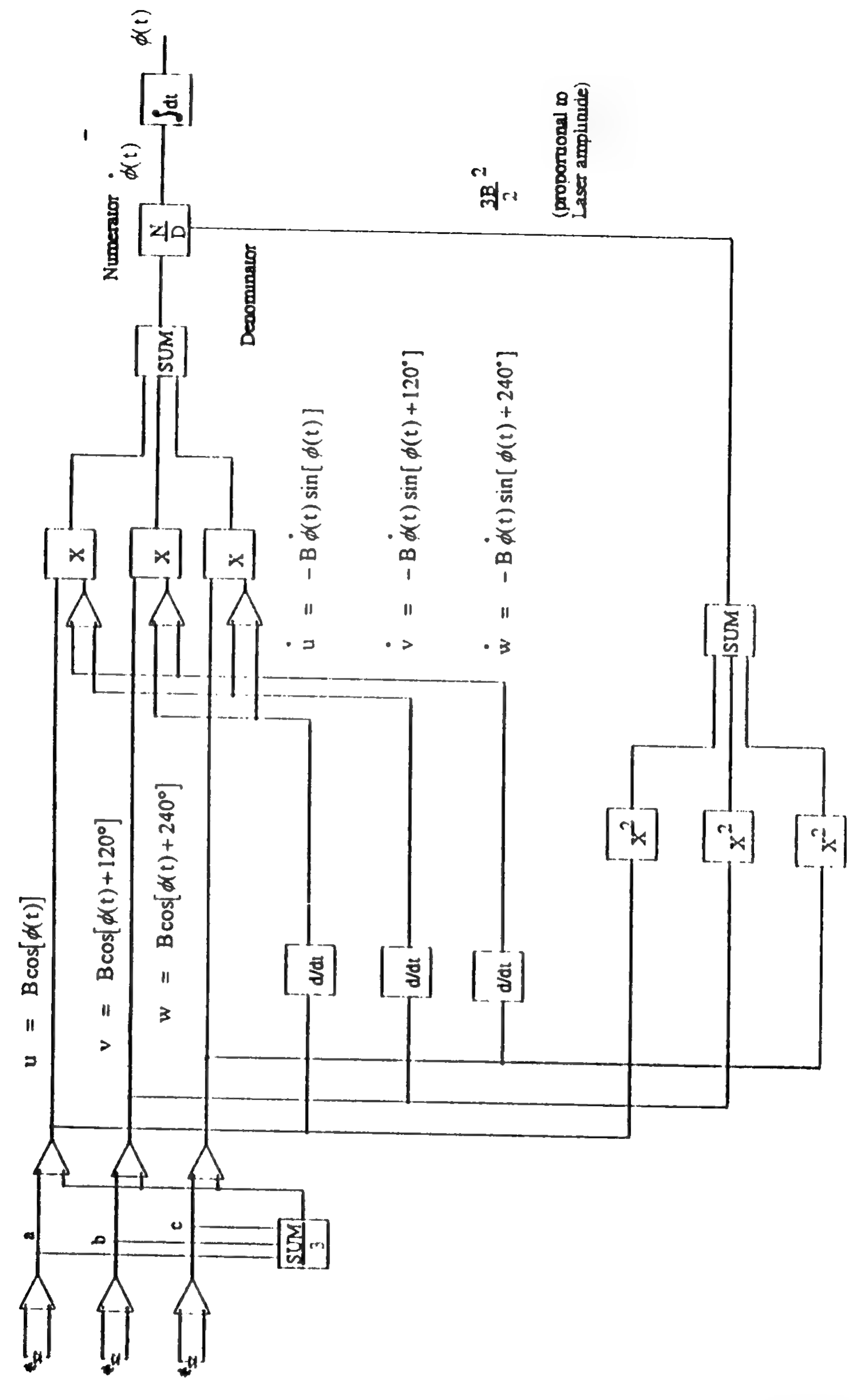
a
b
c
SUM 3
u = Bcos[φ(t)]
v = Bcos[φ(t)+120°]
w = Bcos[φ(t)+240°]
d/dt
d/dt
d/dt
X
X
X
$\dot{u}$ = − B $\dot{\phi}$(t) sin[φ(t)]
$\dot{v}$ = − B $\dot{\phi}$(t) sin[φ(t) +120°]
$\dot{w}$ = − B $\dot{\phi}$(t) sin[φ(t) + 240°]
SUM
Numerator
Denominator
N/D
$\dot{\phi}$(t)
∫dt
φ(t)
X^2
X^2
X^2
SUM
$\frac{3B^2}{2}$
(proportional to Laser amplitude)

Figure 1.2 Block Diagram of the Symmetric Demodulation Algorithm.

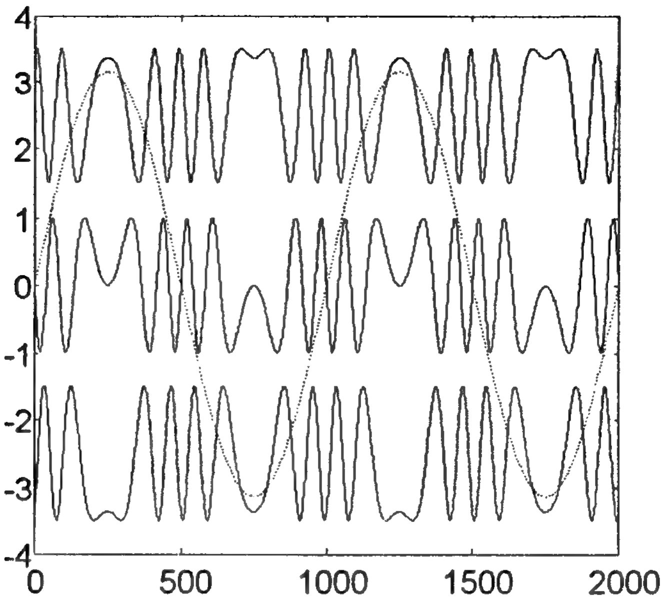

Figure 1.3 Theoretical Plot of 3x3 Coupler Output. Wave Forms a, b and c (solid lines) for an Input Signal (dotted line) which Produced 4 pi Radians of Optical Phase Shift in the Interferometer. The Wave Forms a, b and c have been Displaced from one another for Clarity.

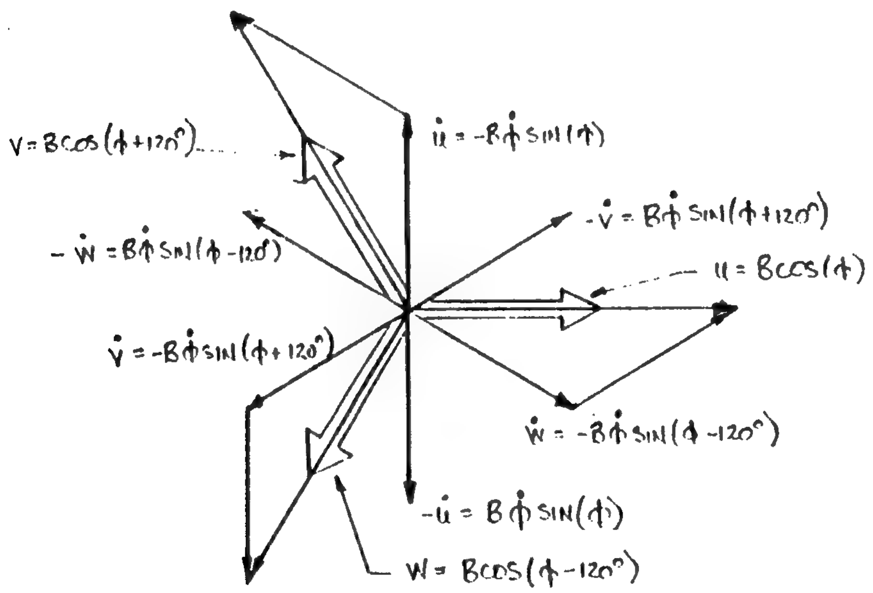
$V = B\cos(\phi + 120°)$
$\dot{u} = -B\dot{\phi}\sin(\phi)$
$-\dot{v} = B\dot{\phi}\sin(\phi + 120°)$
$-\dot{w} = B\dot{\phi}\sin(\phi - 120°)$
$u = B\cos(\phi)$
$\dot{v} = -B\dot{\phi}\sin(\phi + 120°)$
$\dot{w} = -B\dot{\phi}\sin(\phi - 120°)$
$-\dot{u} = B\dot{\phi}\sin(\phi)$
$W = B\cos(\phi - 120°)$

Figure 1.4 Phasor Diagram of Symmetric Demodulation Signals.

2. Symmetric Demodulation

The symmetric demodulation algorithm is is based upon the following expression:

$$\phi(t) \quad \propto \quad \int \frac{u(\dot{v}-\dot{w})+v(\dot{w}-\dot{u})+w(\dot{u}-\dot{v})}{(u^2+v^2+w^2)}\,dt, \tag{1.5}$$

where $\phi(t)$ is the signal of interest to be recovered and the dots above variables indicates a time derivative.

The output intensity of each leg of the interferometer is as previously defined:

$$a \quad = \quad A \quad + \quad B\cos[\phi(t)], \tag{1.6}$$

with similar expressions for b and c. After photo detection and amplification, the three symmetric voltages are summed and scaled by a factor of $-1/3$ thereby producing an estimate of the average signal level. This average dc content is then subtracted from each of the interferometric voltage signals leaving only AC fluctuations about a bias of zero volts. The interferometric signals, with the average dc component removed are now denoted by u, v and w where

$$\begin{aligned} u &= B\cos[\phi(t)]; \\ v &= B\cos[\phi(t)+120°]; \text{ and} \\ w &= B\cos[\phi(t)+240°]. \end{aligned} \tag{1.7}$$

The three signals u, v and w are then differentiated to produce the derivative signals

$$\begin{aligned} \dot{u} &= -B\,\dot{\phi}(t)\sin[\,\phi(t)\,]; \\ \dot{v} &= -B\,\dot{\phi}(t)\sin[\,\phi(t)+120°\,]; \text{ and} \\ \dot{w} &= -B\,\dot{\phi}(t)\sin[\,\phi(t)+240°\,]. \end{aligned} \tag{1.8}$$

Note that $\dot{\phi}(t)$ has come out of the arguments of the trigonometric functions. These three symmetric-interferometric signals are then cross multiplied by the difference of the "complimentary" derivatives to yield:

$$u(\dot{v}-\dot{w}) = B\cos(\phi(t))\left[-B\dot{\phi}(t)\sin(\phi(t)+120^\circ) + B\dot{\phi}(t)\sin(\phi(t)+240^\circ)\right];$$
$$v(\dot{w}-\dot{u}) = B\cos(\phi(t)+120^\circ)\left[-B\dot{\phi}(t)\sin(\phi(t)+240^\circ) + B\dot{\phi}(t)\sin(\phi(t))\right]; \text{and} \quad (1.9)$$
$$w(\dot{u}-\dot{v}) = B\cos(\phi(t)+240^\circ)\left[-B\dot{\phi}(t)\sin(\phi(t)) + B\dot{\phi}(t)\sin(\phi(t)+120^\circ)\right].$$

Figure 1.4 shows a plot of the signal phasors for the above combination of signals.

It can be shown by trigonometric identities that equation (1.9) can be simplified to:

$$u(\dot{w}-\dot{v}) = \sqrt{3}B^2\dot{\phi}(t)\cos^2(\phi(t));$$
$$v(\dot{u}-\dot{w}) = \sqrt{3}B^2\dot{\phi}(t)\cos^2(\phi(t)+120^\circ); \text{ and} \quad (1.10)$$
$$w(\dot{v}-\dot{u}) = \sqrt{3}B^2\dot{\phi}(t)\cos^2(\phi(t)+240^\circ).$$

By noting:

$$\cos^2(x) + \cos^2(x+120^\circ) + \cos^2(x+240^\circ) = \tfrac{3}{2}, \quad (1.11)$$

the sum of the three equations in (1.10) can be shown to result in:

$$\dot{\phi}(t)B^2\left[\cos^2(\phi(t)) + \cos^2(\phi(t)+120^\circ) + \cos^2(\phi(t)+240^\circ)\right] = \tfrac{3}{2}\sqrt{3}B^2\dot{\phi}(t). \quad (1.12)$$

The factor of B^2 varies as a function of laser intensity, temperature, and the polarization angle of the light in the fiber. This dependence is removed by summing the squares of u, v and w.

$$u^2 + v^2 + w^2 = \tfrac{3}{2}B^2. \quad (1.13)$$

This result is used as a normalizing division factor. The ratio of equation 1.12 to equation 1.13 is then integrated to produce $\phi(t)$.

II. DESIGN OF A SYMMETRIC DIGITAL DEMODULATOR

This chapter discusses the details of the construction of a digital symmetric demodulator based upon the algorithm described in chapter 1. Available for use during this thesis was an IBM AT PC, a digital signal processing (DSP) plug-in board, and an interferometric signal simulator. The DSP card, the PC31, was built by Innovation Integration of Moorepark, California. It is described in Appendix A. The heart of the PC31 is comprised of a TMS320C31 DSP processor [Ref 10], which can be operated with either 32 bit integer precision or full 32 bit floating point precision using extended precision registers. It's 60 nsec single instruction cycle provides 16.7 million instructions per second (MIPS), or 33.3 million floating point operations per second (MFLOPS). The PC31 plugs into any IBM PC expansion bus (ISA), and can be fully programmed using either the Texas Instruments TMS320 Floating-point DSP Optimizing C compiler [Ref 11], or the TMS320 Floating-point DSP Assembly Language Tools [Ref 12].

The Interferometric signal simulator is described in Appendix B. The simulator is capable of simulating interferometric signals with optical phase shifts from several hundred micro-radians to about 2 pi radians.

The symmetric demodulator design involves the following phases: Photo detection of the three optical outputs from a 3x3 coupler; conversion to voltage signals proportional to the light intensity; digitization of the input interferometric signals; digital differentiation, multiplication, division, and integration. For my design, I have assumed that the three optical interferometric signals, each 120° out of phase with each other, have been suitably detected and conditioned to provide analog voltages which are limited to 2.75 Volts peak-to-peak. I now turn my attention to a discussion of each stage of the designed digital demodulator.

A. INITIAL DEMODULATOR DESIGN

1. Signal Digitization.

The PC31 provides two independent channels for analog input. Digital conversion is via one Burr Brown DSP102 ADC chip [Ref 13]. The DSP102 is a two channel analog-to-digital converter, ADC, which can operate at up to 200 kHz sampling rates using either a 16 or 18 bit conversion size. Analog full scale input is set at 5.50 volts peak-to-peak. The DSP102 communicates with the TMS320C31 via a serial peripheral data bus. This data bus can operate at a maximum of 8.33 MHz. The DSP102 is conFigured in the cascade mode. When strobed, it simultaneously converts both channels A and B using a 16 bit conversion word and transmits the concatenated 32 bits across the peripheral serial bus. To convert both channels and transmit the 32 bit result to the TMS320C31 requires 4.8μsec at the maximum possible transport rate of 8.33 MHz. Channel A data occupies the upper 16 bits and channel B data the lower 16 bits of the 32 bit word conversion word. Using 16 bit conversion sizes results in approximately 84μV peak quantization per bit, with an error of one half the least significant bit, or ±42μV.

The first problem faced with the design of the demodulator was how to sample three input channels using only one DSP102. Fortunately the PC31 allowed for eight channels to be multiplexed on the A input of the DSP102. Therefore as part of the first design, two of the three interferometric wave forms would have to be digitized by multiplexing them on channel A of the DSP102. This was the easiest approach to sampling, without designing complicated external digitization circuitry. However, this would reduce the effective sampling rate and would not produce simultaneous samples. Using a multiplexed digitization scheme, sampling channels A first and then B and C together would reduce the maximum possible sampling frequency by a factor of two. This will limit the maximum acceptable fringe rate (optical phase shift in the interferometer times signal frequency) of the demodulator. At the maximum possible sampling rate of 200 kHz, the two conversions required to sample the three interferometric signals would require a minimum time of about 10μsec. This should allow sufficient time to process one complete pass of the demodulation algorithm.

2. Algorithm Considerations.

As a first design, it was decided to calculate the derivatives of the interferometric signals using the method of first difference. This was chosen for its simplicity in implementation and least amount of computational burden. Since the TMS320C31 can support both floating point and integer operations, it was decided to initially implement the algorithm using integer operations since the input to the algorithm would be a 16 bit signed integer from the analog to digital converter, and the output would ultimately be in integer form for taking power spectra. However, careful consideration of the various signal maximums and minimums would be required to avoid overflows.

The denominator term in the algorithm takes the sum of the squares of the three input interferometric wave forms, essentially measuring the total optical power. Three 16 bit signed integers squared and added together could cause overflow in a resulting 32 bit integer. Therefore, the denominator was scaled down by a factor of 2^{16}. Similarly, the output of one of the multipliers is also a 16 bit number squared. This gets added to two other multiplier outputs, resulting in a potential overflow in a 32 bit integer. Therefore, each output from the multipliers was scaled down by a factor of two. Since the denominator had been scaled by 2^{16} the result from the division of the numerator (sum of the three multiplier outputs) and the denominator would be an integer just larger than 2^{16}. The integration of $\dot{\phi}(t)$, was implemented using a first order difference equation $y(n) = 0.99 * y(n-1) + x(n)$, where the initial decay constant was set to 0.99. The output from the integrator is proportional to the signal that produced the original optical phase shift in the interferometer.

With the initial design considerations outlined, it was decided to initially program the algorithm using a well known platform, the IBM PC. Therefore the demodulation algorithm was first written in Turbo C [Ref 14] and tested on an IBM PC. However, this testing only used computer generated interferometric wave forms. The algorithm was also programmed in MATLAB [Ref 15]. Here, testing was first conducted using computer generated interferometric wave forms as inputs and then using data samples from the interferometric simulators taken using the PC31 and multiplexed sampling. The results of these simulations are described below.

B. SIMULATION ON AN IBM PC USING TURBO C

The designed demodulation algorithm was written in Turbo C and tested on an IBM PC. The demodulation program, TCDEMOD.C can be found in Appendix C. It was decided that by using a well known programming language and host computer, it would be easier to test and evaluate the demodulation algorithm from within a controlled environment. The MATLAB programming environment was used to display output from the TCDEMOD.C program. The MATLAB script file, PLOTDAT.M, used to display the data, is also included in Appendix C. The interferometric wave forms were generated by the TCDEMOD.C program and then demodulated using the algorithm discussed above.

Figures 2.1 - 2.4 show the various signals at different stages throughout the demodulation process. Computer generated inteferometric waveforms produced by a single sinusoid at 1/128th the sampling frequency with an amplitude which caused pi radians of optical phase shift in the interferometer were used as input for the demodualtion algorithm. The simulated interferometric signals were created at full quantization of $\pm 2^{15}$. Figure 2.1 shows a plot of 200 samples of the computer generated interferometric input signals for full 16 bit quantization (±32768). Figure 2.2 shows three plots of the derivatives of each input signal after the dc bias has been removed. The derivatives were calculated using a first difference approach. From the plots it can be seen that the maximum range for the derivative signals is ±5000 for this particular input optical phase shift of pi radians. Figure 2.3 shows a plot of the first 500 samples of the numerator (left) and denominator (right) output from the demodulation algorithm. The numerator signal is proportional to $\dot{\phi}(t)$, which is the derivative of the signal of interest. Notice that it has been represented as 32 bit integer ($\pm 2.147\text{x}10^9$). The denominator term, which is an estimate of the input laser power, is used to compensate for variations in the input signal caused by laser fluctuations. It is calculated by summing the squares of the signals u, v, and w and has been scaled down to accommodate a 16 bit representation. Notice the quantization noise associated with this scaling. The denominator fluctuates between 2.3435 and 2.3437, 2 bits of quantization noise, about 160 μV at full scale. Finally Figure 2.4 displays the output of the algorithm. This signal is from the digital integrator, a simple difference equation $y(n) = 0.99y(n-1) + x(n)$. Since this program only ran for just over 1000 samples the integrator decay is quite noticeable. The transients present at the first of the output are due to improper algorithm initialization.

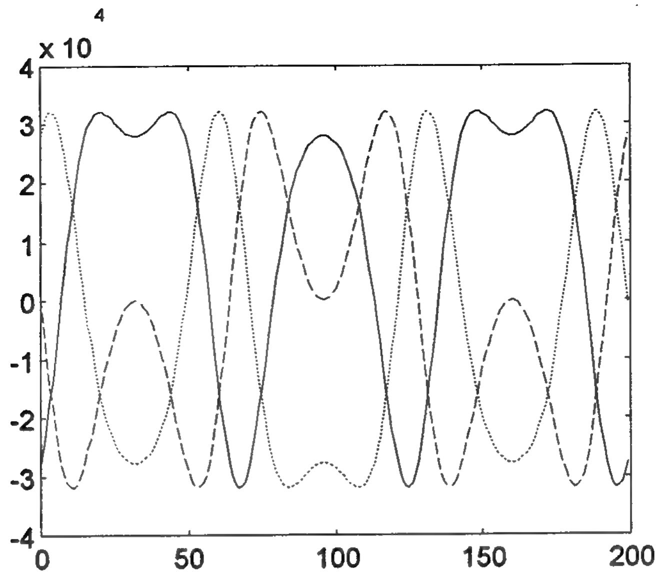

Figure 2.1 Input Interferometric Signals at Full Quantization. Signal a (solid line) Signal b (large dotted line) and Signal c (small dotted line).

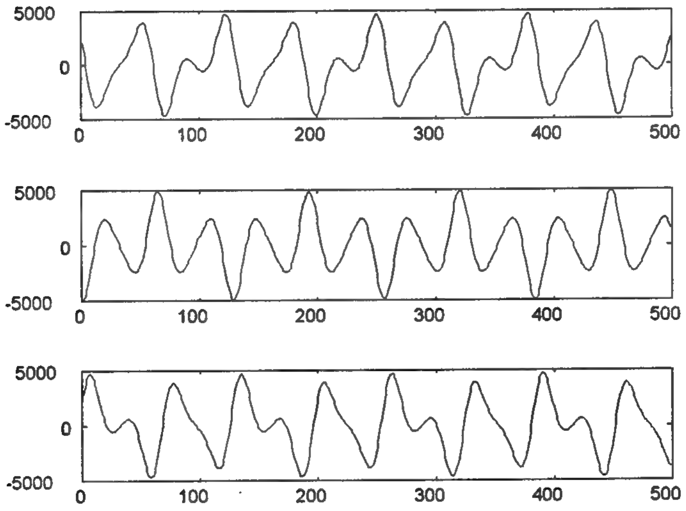

Figure 2.2 Derivatives of the Interferometric Inputs after the dc Component has been removed. Derivative of Signal u (top) Derivative of Signal v (middle), and Derivative of Signal w (bottom).

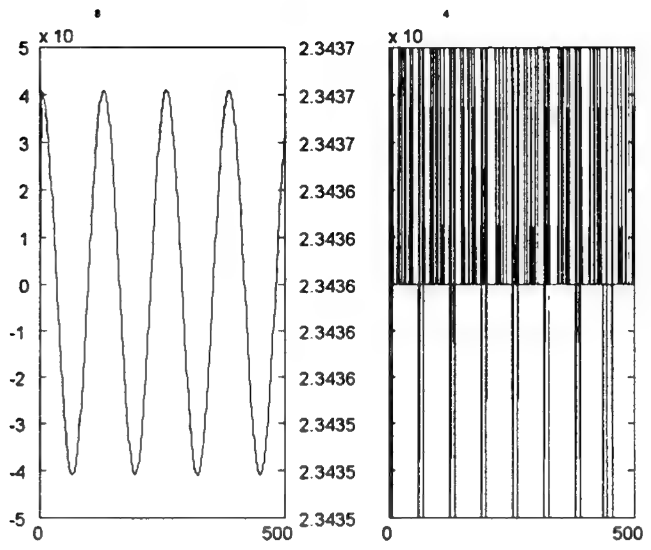

Figure 2.3 Output from the Numerator Calculations (left) and Output from the Denominator Calculations (right).

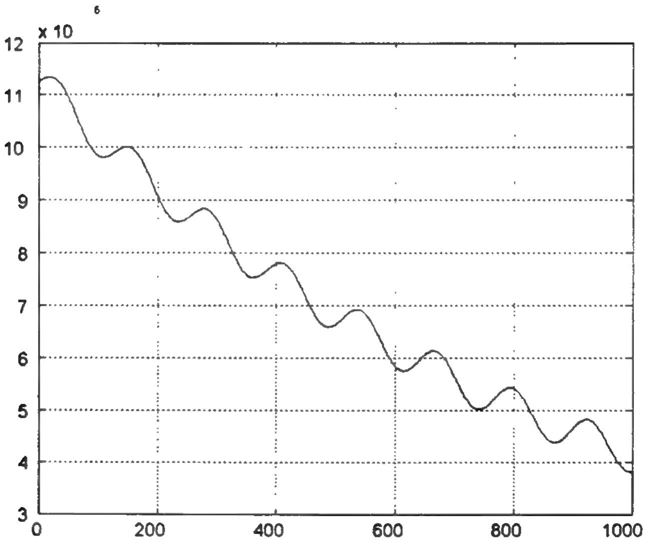

Figure 2.4 Demodulation Output which is Proportional to $\phi(t)$

The next set of Figures, 2.5 - 2.8, were again produced using the Turbo C demodulation program, TCDEMOD.C and MATLAB script file PLOTDAT.M. This time the program demodulated 10,000 points of computer generated interferometric data. The input signals and the last 2048 points of the output signal were stored and used for plotting in MATLAB. Figure 2.5 shows 200 points of the simulated interferometric wave forms for an input sinusoid at a frequency of 1/128 that of the sampling frequency and an amplitude which would generate 10 radians of optical phase shift in an interferometer. Figure 2.6 shows plots of the resulting output from the demodulation process and its resulting power spectrum. Since only the last 2048 of the 10,000 output samples generated were plotted, the initial transients in the integrator have decayed and are not present in the ouput. The power spectrum plot was produced using the PSD.M script file from MATLAB's signal processing toolbox, version 3.1. The DFT size was 2048 points and a Hanning window of the same size was used to window the data. Notice that the signal has no dc present and it is outside the range of 16 bit quantization. The frequency of the signal is at 781.3 Hz, 1/128 of 100 kHz sampling rate. The signal-to-noise ratio (SNR) is 140.6 dB, the peak output occurring at 114.8 dB. Notice the low distortion present. For a given frequency, as the optical phase shift in the interferometer increases, a result of an increase in the measured acoustic field, the distortion of the output increases. The next two Figures show this effect. Figure 2.7 shows a plot of the interferometric wave forms generated for an input sinusoid, again at a frequency of 1/128 of the sample frequency, fs, but at a much greater acoustic amplitude (50 radians optical phase shift produced in the interferometer). Figure 2.8 shows a plot of the output of the demodulator and a plot of its power spectrum. The effective fringe rate (optical phase shift times signal frequency) is now 39,000 fringes per second. Notice however that even at large distortion levels the output signal still resembles a sinusoid and may still be reasonably detected.

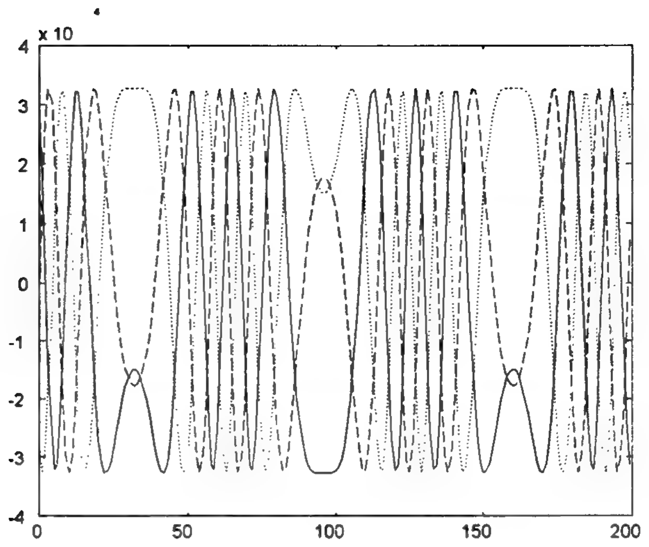

Figure 2.5 Computer Generated Interferometric Signals. The Input Signal is a Sinusoid at 1/128 of the Sample Frequency, fs with an Amplitude which Generated 10 Radians of Optical Phase Shift in the Interferometer.

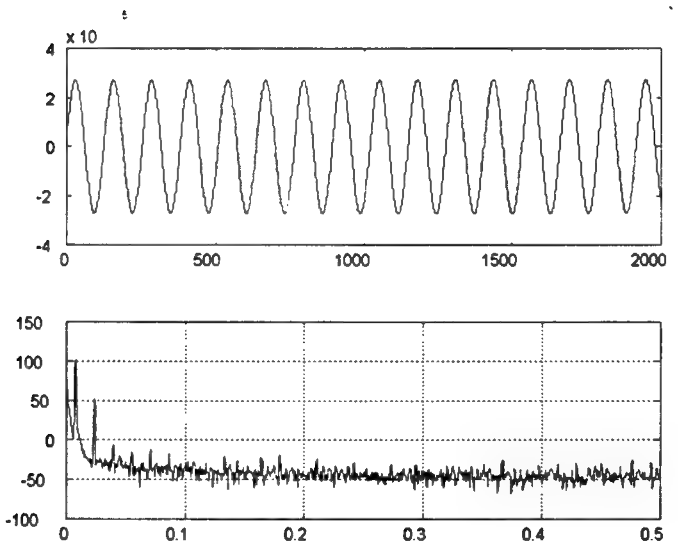

Figure 2.6 2000 Points of the Demodulator Output (top), and its Power Spectral Density Plotted in dB's (bottom).

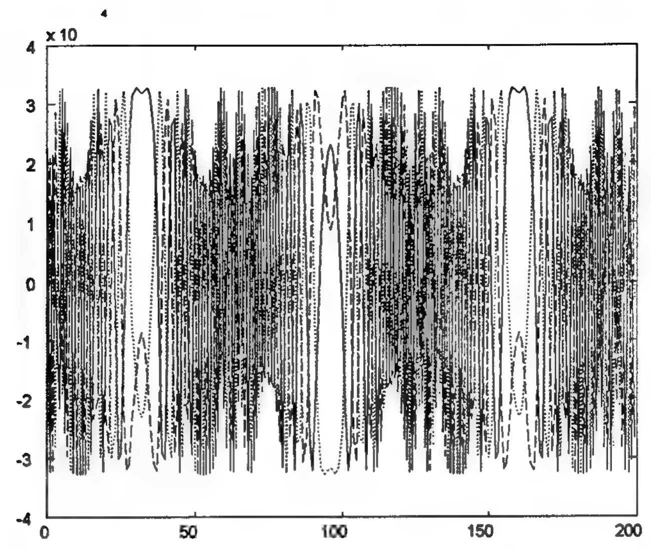

Figure 2.7 Input Interferometric Wave Forms for a Sinusoidal Input at 1/128 the Sample Frequency, fs and an Amplitude which Generated 100 Radians of Optical Phase Shift in the Interferometer.

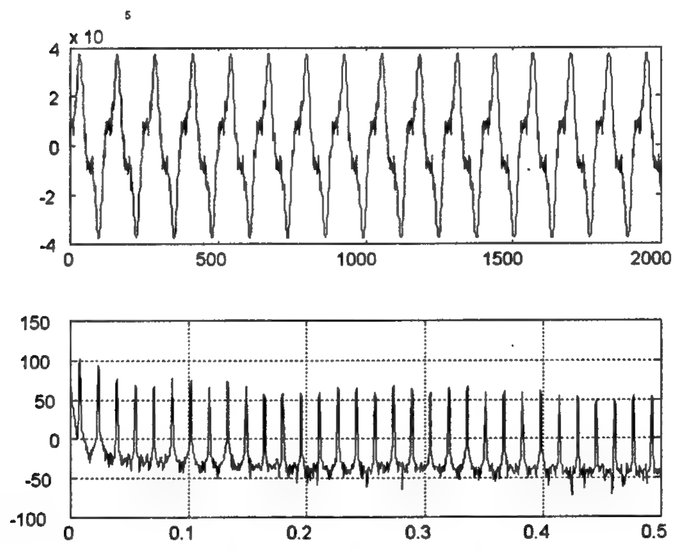

Figure 2.8 2000 Points of the Demodulator Output (top) and its Power Spectral Density Plotted in dB's (bottom).

C. SIMULATION USING MATLAB

It was decided to also simulate the demodulation algorithm in MATLAB. MATLAB offers a very easy programming environment were signals and noise can be readily simulated, processed and displayed. It offers the ideal environment for investigating the behaviour of the algorithm. MATLAB uses floating point arithmetic for its internal calculations, but output data can be rounded or fixed to the nearest integer. The script file, DEMOD.M, written for the MATLAB simulation can be found in Appendix C.

Figures 2.9 - 2.13 display various output wave forms from the demodulation process. Figure 2.9 is a plot of the MATLAB generated simulated interferometric signals. The input signal was a sinusoid at 600 Hz with an acoustic amplitude which resulted in pi radians of optical phase shift in the interferometer. The simulated signals had a dc offset of 0.5 Volts peak and quantization amplitude equivalent to 2^{14}. Figure 2.10 is a plot of the derivatives of the three interferometric signals u, v and w after the dc component has been removed. The upper plot shows $\dot{u}$; $\dot{v}$ is plotted in the middle plot and $\dot{w}$ is plotted in the bottom. The plots of the derivatives closely match those shown for the Turbo C program. Figure 2.11 shows plots of the outputs from the three multiplier stages. The top plot shows the output multiplication of signal u with the complementary derivatives $(\dot{w}-\dot{v})$, the middle plot shows the output from the multiplication of v with $(\dot{u}-\dot{w})$, and the bottom plot displays the output from the multiplication of w with $(\dot{v}-\dot{u})$. Figure 2.12 displays plots of the numerator and denominator signals. Notice how the denominator once again has one bit of quantization noise. This is a result of fixing the floating point value output from MATLAB. The denominator is very close to a constant value for this simulation. However, the program allows for simulation of slowly fluctuating laser inputs. Notice also that at this level of optical phase shift the numerator term has not yet exceeded 32 bit integer quantization limit. Figure 2.13 shows the output signal from the demodulation process and its power spectral density. The output looks very clean, and indeed has very little distortion as indicated by the power spectrum. Notice that the output from this MATLAB simulation is less distorted than the output from the Turbo C simulations. The demodulation algorithm was extensively simulated using MATLAB, over a wide range of frequency ratios (input frequency divided by sampling frequency), an input optical phase shifts. Several observations were noted. First, the numerator overflows a 32 bit signed representation using integer arithmetic for large fringe rates (fringe rates greater than 100). Therefore the components making up the numerator will have to be scaled by a factor of one half. This will be sufficient to avoid any overflow. Secondly, at small optical phase shifts, 100's of micro radians, the demodulator sensitivity will be determined by the signal derivatives which under flow 16 bit quantization first. With these discoveries in mind it was now time to implement the algorithm on the PC31.

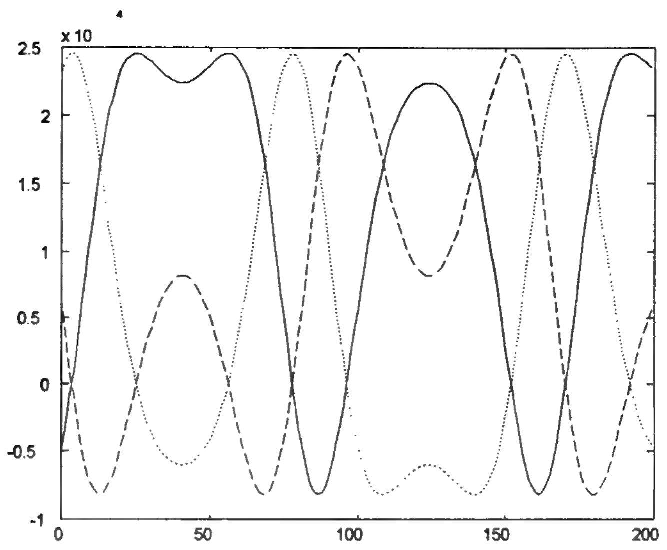

Figure 2.9 MATLAB Generated Interferometric Wave Forms. Signal a (dotted line), Signal b (dashed line) and Signal c (solid line).

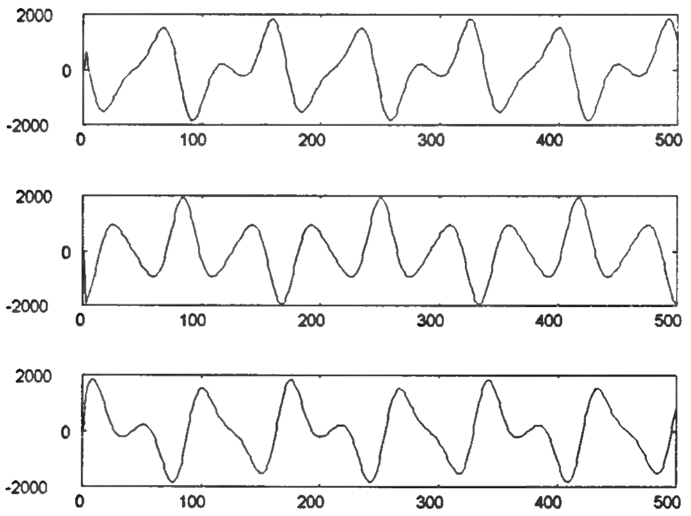

Figure 2.10 Derivatives of the Interferometric Signals with the dc Component Removed. Derivative of signal u (Top), Derivative of signal v (Center), and Derivative of Signal w (Bottom).

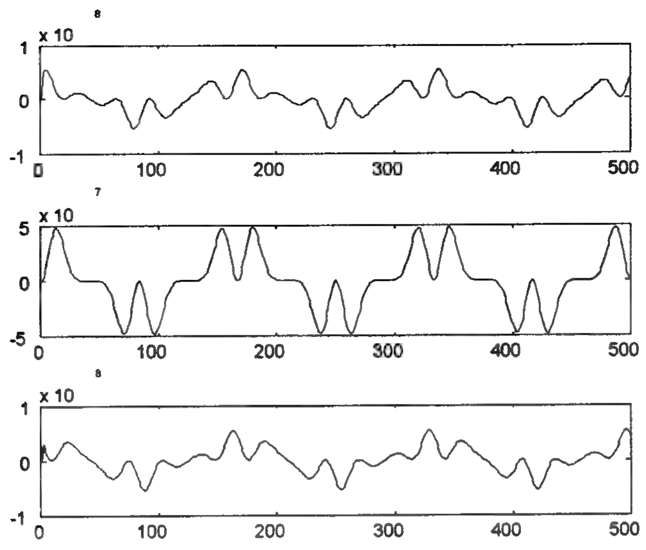

Figure 2.11 Outputs from the Three Multiplier Stages. (Top) $u(\dot{w} - \dot{v})$, (Center) $v(\dot{u} - \dot{w})$, and (Bottom) $w(\dot{v} - \dot{u})$.

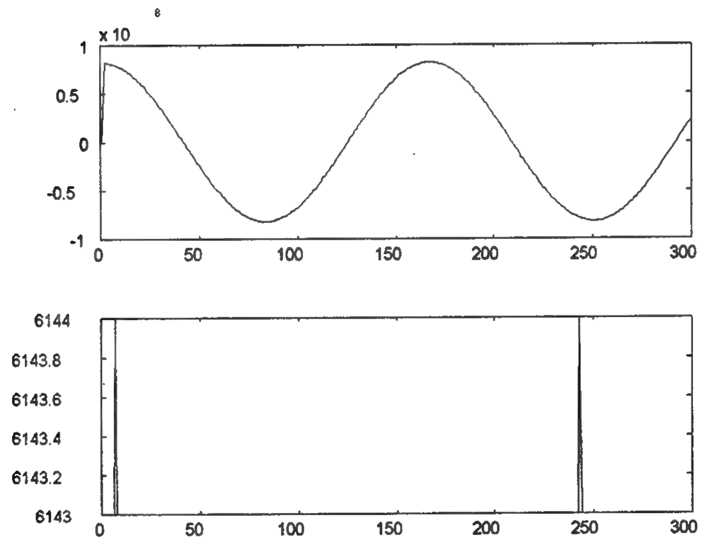

Figure 2.12 Numerator signal (Upper plot). Denominator signal (Lower plot).

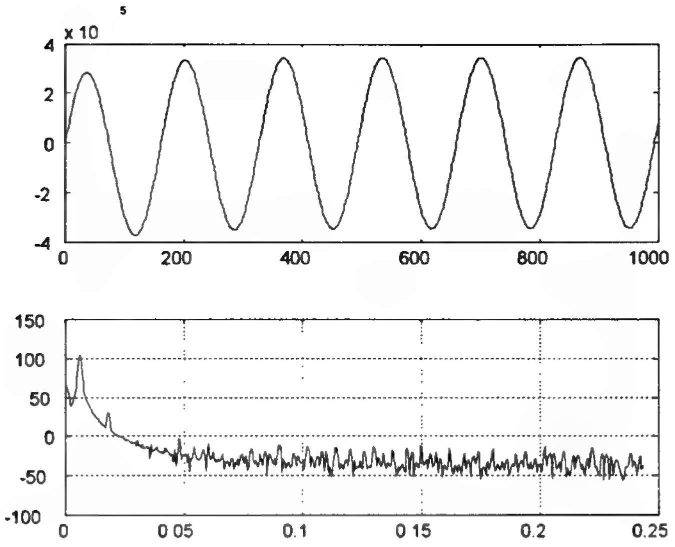

Figure 2.13 Demodulator Output (Upper plot) and its Power Spectral Density Plotted in dB's vs. Nyquist Frequency (Lower plot).

D. PROGRAMMING THE PC31

Once the algorithm had been successfully programmed and tested in MATLAB, it was time to consider an implementation on the PC31 DSP board. I initially decided to program the demodulation algorithm in C using the TMS320 optimizing C compiler. The program titled TMSDEMOD.C can be found in Appendix C. The most challenging part of implementing the demodulator design on the PC31 was in choosing the method used to sample the three interferometric signals.

1. TMS320 Optimizing C Compiler Programs

The DSP102 analog-to-digital converter (ADC) can be strobed to start a conversion by either writing to a memory location, or by jumping one of three different programmable clocks on the PC31 board to the convert pin of the DSP102. Once a conversion is complete (analog-to-digital conversion plus transport of 32 bits along the serial port), the TMS320C31's internal serial port generates an interrupt signal when its receive buffer is full. This interrupt signal can be used to interrupt the CPU at the end of a conversion. It can also be masked, in which case the program must poll the serial port and wait for the conversion to complete. I initially decided to run the conversion process using a polling approach. The conversion process of the multiplexed input line IN0 starts when a dummy store was made to the memory location decoded for the ADC. After receiving the convert command the DSP102's internal hold circuitry keeps the inputs constant while the conversion is in process. This allows the inputs to the DSP102 to be changed immediately after a convert command has been given. Therefore while the conversion is in process the multiplexer channel is switched to another input, IN1. This gives the multiplexer sufficient time to settle (the analog multiplexer requires 1 μsec for its output to settle to within 0.1% of true value. Waiting 3.5 μsec increases the accuracy to 0.01%). After 4.8 μsec, the converted data is received at the serial port and can be read by the program. The first conversion made by the DSP102 is not valid and must be discarded. The next conversion represents the first data sample. Once the data is read, a new conversion can be signaled with a write to the ADC's memory location and the multiplexer can be switched back to channel IN0. The data read from the DSP102 is shifted to the right 16 bits, representing signal sample a, which is stored until all three signal samples are read. The program then waits for the next sample by polling the status of the serial port. Once the next conversion has been received it is read and the 32 bit conversion word is separated into sample b (upper 16 bits) and sample c (lower 16 bits). Once the three samples have been read the program enters the demodulation process. The denominator term, derivatives and numerator are calculated and used to compute the first output sample. This loop repeats itself until 4000 data samples have been computed.

Several C programs were written to test various aspects of the demodulation algorithm and to save other intermediate variables. The internal memory storage of the PC31 was limited to 26,600 32 bit words of storage for both program and data. Therefore not all intermediate data could be stored at one time. Once computed the data could be downloaded from the PC 31 for viewing and manipulation in MATLAB. Although the programs written in C and compiled using the TMS320 optimizing C compiler were successful in implementing the demodulation algorithm, the execution was not very fast and this limited the effective sampling rate. This was due in part to the optimizing C compilers usage of the TMS320C31 register set and the adherence to the TMS C function calling conventions. Also, the routines were slowed down because of the need to wait for ADC conversions while polling. The solution to the two problems was to switch to assembly language programming, instead of using the C compiler, and use serial port interrupts instead of polling. Use of interrupts would allow the demodulation routine to continue with computations instead of having to poll the serial port for new ADC conversions.

2. TMS320 Assembler Programs

After several long months of learning the intricacies of the TMS320C31 addressing modes, I was ready to tackle the job of writing the demodulation algorithm in assembly language. I first wrote a simple assembly language routine to sample data presented to channels IN0 and IN1 of the multiplexer and input B of the DSP102. This program is called TST3.ASM and it is included in Appendix C. The program takes up to 8192 points of data from the three input channels. The DSP102 ADC is driven using internal timer clock number one (TCLK1) from the TMS320C31. The serial port interrupts the CPU when it has received the full conversion from the ADC. Input A to the DSP102 is multiplexed through channels IN0 and IN1 of the multiplexer. Interferometric signal a is connected to IN0 and interferometric signal b is connected to IN1. Interferometric signal c is connected directly to input B of the DSP102, which is not multiplexed. After initialization of the serial port, internal timer and program variables, the interrupts are enabled and the program waits in an infinite loop until interrupted. Data from signal a (upper 16 bits of first serial port read) is read and stored after the first interrupt. The data read from input B of the DSP102 is discarded. The multiplexer channel is then switched to channel IN1 and the routine waits for the next interrupt. Once interrupted, data from signal b (upper 16 bits) and signal c (lower 16 bits) are read and stored. This completes the sampling of the first three interferometric signals. The multiplexer is then switched back to channel IN0 and waits for the next interrupt in order to get the next set of samples. A data index pointer increments after collecting the three data samples. After taking 8192 samples the interrupts are disabled and the PC31 can be reset. The sampled data can then be downloaded to the host computer. This routine can be run at the maximum sampling frequency of 200 kHz. However, because of the need to multiplex the input data, the effective sampling rate for all three channels is reduced in half to 100 kHz.

Once the multiplexed data sampling routine, TST3.ASM had been written and tested, it was time to consider the implementation of the complete demodulation algorithm. Program TST5.ASM, shown in Appendix C, is the routine I developed to implement the demodulation algorithm on the PC31 using integer operations. This routine accepts three simulated interferometric signals a, b and c each 120° out of phase with each other. The outputs from the interferometric signal simulators are applied to IN0, IN1 and input B on the DB37 analog connector on the back of the PC31. The algorithm digitizes this data and implements a symmetric demodulation of the input signals using integer operations. 8192 points of the denominator, numerator and output are computed and stored for downloading to the host computer. Several interesting statistics about the algorithms timing were clarified during the development of this routine. First, it takes about 540 nsec (9 instructions) from an interrupt occurring to the first instruction of the interrupt service routine. Secondly, the time required to read sample a after an interrupt has occurred, switch the multiplexer to channel IN1, store the data and return from the interrupt takes about 1.5 μsec. Once the first data sample has been read, from the time the second interrupt is generated to the end of the demodulation process requires 10 μsec. However, this time varies by as much as 1 μsec. This is because of the integer division. Each integer division requires between 31 - 62 cycles to execute depending upon the amount of normalization needed. However, each floating point divide requires only a fixed 40 cycles to execute. The assembly language routines which implement these divisions were provided with the TMS320 development package.

Since using a floating point divide may be quicker in some cases than an integer divide, I decided to also implement the demodulation algorithm using floating point operations. Assembly language routine TST7.ASM implements the demodulation algorithm floating point operations on the PC31. This program is also included in Appendix C. Only minor changes were required to TST5.ASM to implement the algorithm in floating point. However, I had to be careful to use the 40 bit extended precision registers R0 - R7, which offer a full 32 bit mantissa field and an eight bit exponent field. Again 8192 points of the denominator, numerator and output are calculated and stored. Before the results are stored they are converted to 32 bit integers. This was required since the 40 bit extended precision registers cannot be fully stored in a 32 bit memory location. Also, 32 bit integers were required for downloading to MATLAB for analysis.

III. PERFORMANCE OF THE DIGITAL DEMODULATOR

The digital demodulator was tested using the analog interferometric signal simulators as the source for the interferometric wave forms. See Appendix A for a description of the signal simulators. Each simulator was set to output 3.0 volts peak-to-peak. This corresponded to the design input of 1.5 volts peak for each channel of the DSP102. The interferometric simulators were driven using a Stanford Research Systems, Model DS345 function generator which provided the source signal $\phi(t)$. The frequency and amplitude of the input source signal determine the frequency and optical phase shift of the interferometric waveforms. The static phase shift of each simulator was set such that each of the three outputs was 120° out of phase from each other. The testing of the demodulator proceeded in three phases. First, the DSP102 and the multiplexed sampling dynamics were tested for full scale voltage linearity, noise and offset. Secondly, the analog signal simulators were sampled to see how they compared to theoretical computer generated interferometric wave forms. Finally, the demodulator output was tested using several sinusoids at various amplitudes.

A. TESTING OF THE INPUT SAMPLING CIRCUITS.

First the DSP102's offset and gain adjustment were checked using the procedures outlined in the PC31 user's guide [Ref 16:p 40]. The program TST3.ASM, shown in Appendix C, was assembled using the TMS320 assembler and then downloaded to the PC31 to run the data collection program. The multiplexer and ADC were then tested using a single sinusoid applied to IN0, and IN1 of the multiplexer and channel B of the DSP102. A 600 Hz sinusoid was generated using a Stanford Research Systems Model DS345 function generator set to 2.0 volts r.m.s., the full scale voltage for the DSP102 inputs. The three channels were then sampled at 151.5 kHz for an effective multiplexed sampling rate of 75,757.6 Hz. Data was collected on the PC31 and then was downloaded to the PC for analysis using MATLAB.

1. Data for Full Scale Inputs

Figures 3.1 to 3.3 were generated using the MATLAB script file PLOTABC.M, included in Appendix C. The power spectral density estimates were calculated and plotted using the progam PSD.M from the signal processing toolbox, version 3.1. 8192 points were used for the DFT calculations and the data was windowed with a Hanning window of the same size. The Total Harmonic Distortion (THD) estimate was made using the fundamental and the first 10 harmonics. Figure 3.1 shows a plot of the data taken from channel IN0 of the multiplexer (upper plot). The power spectral density of channel IN0 and an estimate of the distortion present is shown in the lower plot. Figure 3.2 shows plots of data taken from channel IN1 of the multiplexer. Theoretically Figures 3.1 and 3.2 should be identical. Any differences between the two plots are due to the differences in the two multiplexer channels. As can be seen from the plots, both channels are very close to each other. Figure 3.3 shows data taken from channel B of the DSP102. In all three plots the 2.0 volts r.m.s. input produces close to full scale readings from all three channels. The signal generator used to generate the test sinusoid or the DSP102 has about 2% total harmonic distortion. This was the same function generator which was used to drive the three interferometric signal simulators.

2. Data for Inputs Grounded.

Next, IN0, IN1 and input B were grounded to the analog ground plane. Program TST3.ASM was downloaded to the PC31 and the three channels were sampled at an effective rate of 75,757.6 Hz. 8192 points of the sampled input were taken, stored and downloaded to the PC for plotting in MATLAB. The plots were generated using the script file PLOTABC.M, however, this time only 4096 points of the data were used for the power spectral density plots resulting in a bin width of 18.5 Hz. Using a 4096 point Hanning data window gave an equivalent noise bandwidth of 22.75 Hz [Ref 17] for the power spectrum plots. Figures 3.4 - 3.6 show plots of this grounded condition. Notice the noise on all three channels. It is digital in nature and has large fluctuations about zero. Figure 3.4 shows sampled data from channel IN0. The upper plot shows 500 points of the sampled data. Using the calculated standard deviation of 3.663 the noise for channel IN0 was calculated to be about 0.3 mV. Figure 3.4 shows plots of data taken from input IN1 to the multiplexer. Using the calculated standard deviation of the data, the noise for this channel was calculated to be also about 0.3 mV. Figure 3.6 shows plots of data taken from input B to the DSP102. Using the calculated standard deviation, the noise for this channel was calculated to be about 0.4 mV. Input B to the DSP102 is slightly more noisy than channel A. This can also be seen from Figure 3.6 where it is easy to see that the gain setting for input B of the DSP102 is slightly higher than that of input A. Also, both channels are slightly offset from zero.

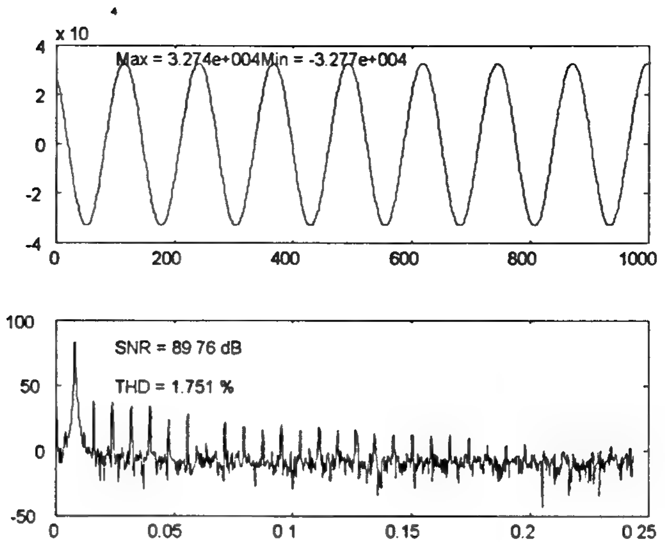

Figure 3.1 Plots of Full Scale Input to Multiplexed Channel IN0. Sampled Data Plotted as a Function of Magnitude and Sample Number (Upper plot). Power Spectrum of Input Plotted in dB's vs. Nyquist Frequency (Lower plot).

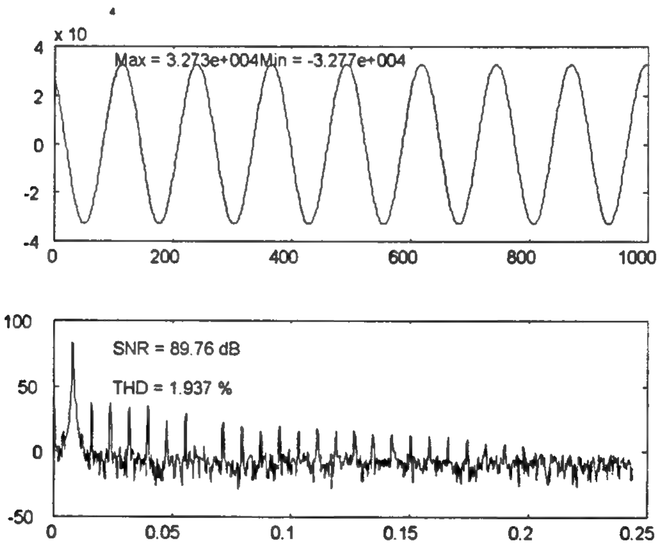

Figure 3.2 Plots of Full Scale Input to Multiplexed Channel IN1. Sampled Data as a Function of Sample Number and Magnitude (Upper plot). Power Spectrum of Input Data Plotted in dB's vs. Nyquist Frequency (Lower plot).

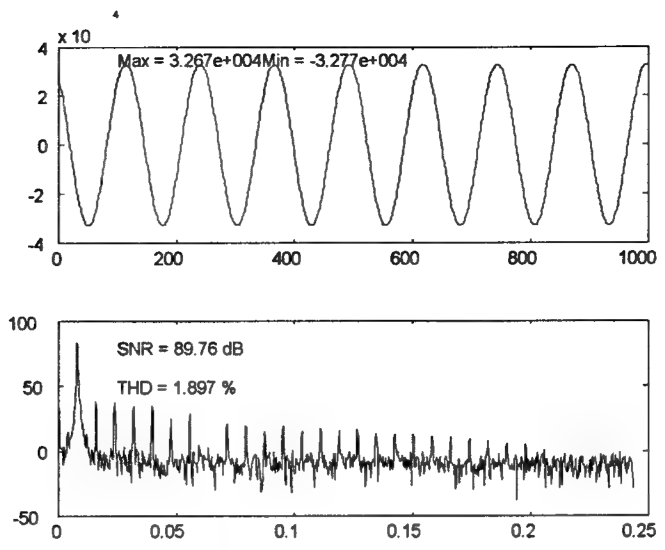

Figure 3.3 Plots of Full Scale Input to Input B of the DSP102. Sampled Data Plotted as a Function of Magnitude and Sample Number (Upper plot). Power Spectrum of the Sampled Data Plotted in dB's vs. Nyquist Frequency (Lower plot).

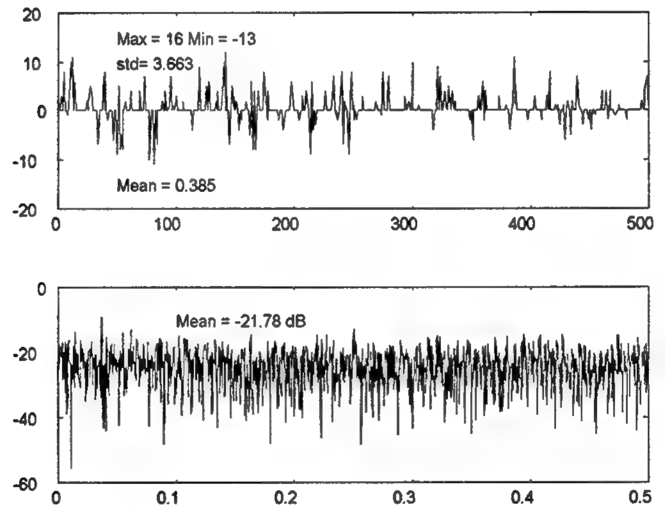

Figure 3.4 Plots of Channel IN0 of the Multiplexer for Grounded Input. First 500 Data Points of Sampled Input (Upper plot). Power Spectral Density of Channel IN0 Plotted in dB's vs. Nyquist Frequency (Lower plot).

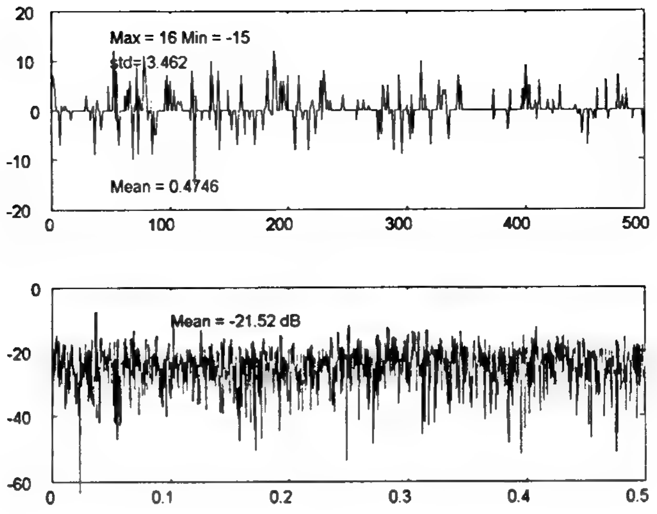

Figure 3.5 Plots of input IN1 of the multiplexer for Grounded Input. First 500 Data Points of Sampled Input IN1 (Upper plot). Power Spectral Density of Grounded Channel IN1 Plotted in dB's vs. Nyquist Frequency (Lower plot).

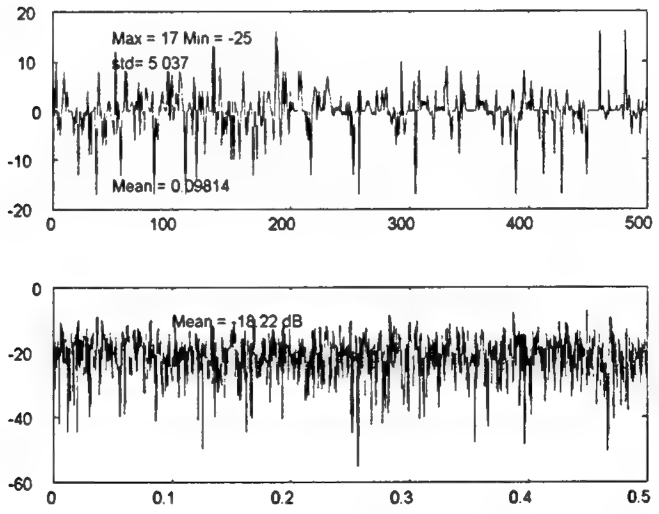

Figure 3.6 Plots of Input B to the DSP102 for Grounded Input. First 500 Data Points of Sampled Input (Upper plot). Power Spectral Density of Grounded Channel B Plotted in dB's vs. Nyquist Frequency (Lower plot).

B. SAMPLING OF THE INTERFEROMETRIC SIGNAL SIMULATORS

The next test was conducted using the interferometric signal simulators. The simulators were driven with a 1.0 kHz sinusoid at an amplitude of 1.0 Volt peak. This produced interferometric wave forms with about 1.0 radian of optical phase shift since the scale factor for the simulators was 0.933 rad/Volt (see Appendix B). The data was taken using program TST3.ASM set to sample the data at an effective sampling rate of 75,757.6 Hz. The output from the PC31 was downloaded, analyzed and plotted in MATLAB using the script file LDTST3.M. This program is included in Appendix C. The power spectral density plots of the interferometric channels were calculated with program PSD.M, using 2048 point DFTs and a Hanning data window of the same size. This gave an equivalent noise bandwidth of 45.5 Hz.

Figures 3.7 - 3.9, show plots of the sampled interferometric channels. Plot 3.7 is a plot of interferometric simulator #1 which was set to a static phase shift of -120°. This was applied to channel IN0 on the analog connector to the PC31. The top plot displays the sampled data while the bottom plot displays the power spectrum plotted in dB's vs. digital frequency f/fs. Figure 3.8 is a plot of interferometric simulator #2 set to a static phase shift of 0. It was connected to IN1 on the analog connector to the PC31. Figure 3.9 shows plots of interferometric simulator #3, set to a static phase shift of +120°. It was connected to Input B of the DSP102. All three simulated wave forms produce spectrums which are typical for interferometric signals.

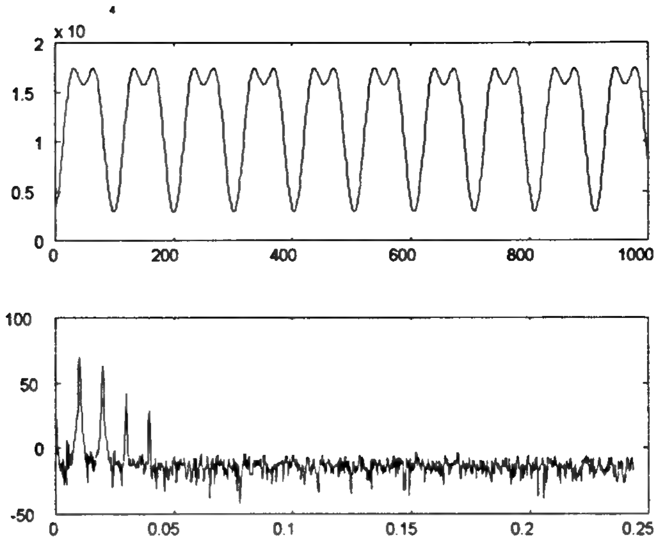

Figure 3.7 Plots of Interferometric Signal Simulator #1 Set to a Static Phase Shift of -120°, 1.0 kHz Input and 1.0 Radian of Optical Phase Shift. Multiplexed Channel IN0 Sampled at 75,757.6 Hz (Upper plot). Power Spectrum of Interferometric Signal Plotted in dB's vs. Nyquist Frequency (Lower plot).

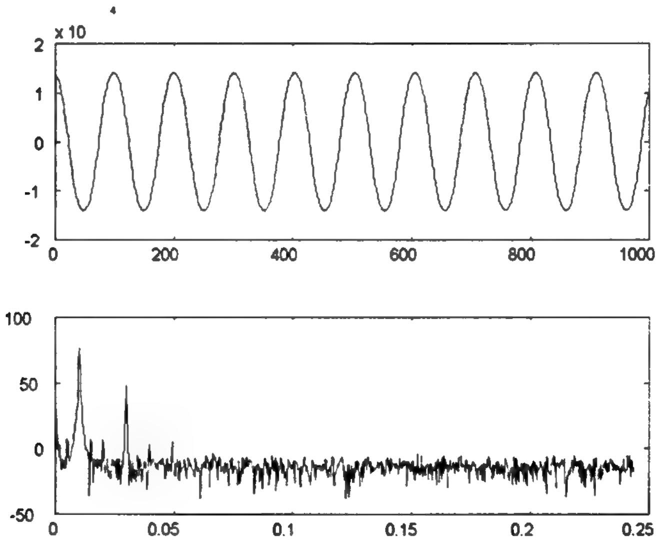

Figure 3.8 Plots of Interferometric Signal Simulator #2 Set to a Static Phase Shift of 0° 1.0 kHz Input and 1.0 Radian of Optical Phase Shift. Multiplexed Channel IN1 Sampled at 75,757.6 Hz (Upper plot). Power Spectrum of Interferometric Signal Plotted in dB's vs. Nyquist Frequency (Lower plot).

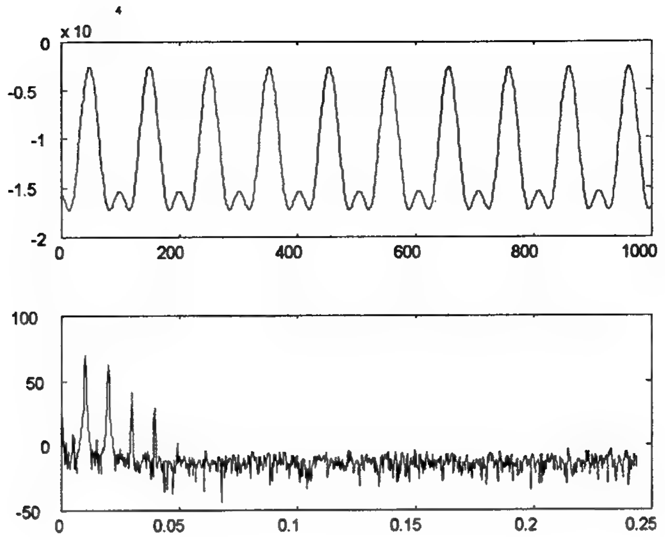

Figure 3.9 Plots of Interferometric Signal Simulator #3 Set to a Static Phase Shift of +120°, 1.0 kHz Input and 1.0 Radian of Optical Phase Shift. Input B to DSP102 Sampled at 75,757.6 Hz (Upper plot). Power Spectrum of Interferometric Signal Plotted in dB's vs. Nyquist Frequency (Lower plot).

C. DIGITAL DEMODULATOR OUTPUT.

The two assembly language programs written to perform the digital demodulation, TST5.ASM (integer operations), and TST7.ASM (floating point operations), were tested. TST5.ASM was run at an effective sampling rate of 69,444.4 Hz, while TST7.ASM could be run a little faster at 75,757.6 Hz. Three interferometric signal simulators driven by the Stanford Research Systems DS345 function generator were used to simulate the interferometric wave forms for the demodulator.

1. Demodulator Response to Input Frequency Variations.

The demodulator response to frequency variation was tested by simulating interferometric signals at constant optical phase shift but at two separate frequencies of 600 and 1200 Hz. For program TST7.ASM using a sampling rate of 75,757.6 Hz the input frequencies were 0.0079 and 0.0158 times the sampling rate, fs. For program TST5.ASM using a sampling rate of 69,444.4 Hz, the input frequencies were 0.0086fs and 0.0173 times the sampling rate, fs. Interferometric wave forms with pi radians of optical phase shift were produced using the interferometric simulators. In each run, 8192 points of the denominator, numerator and output signals were calculated by the demodulation program and later downloaded to the PC. Plotting and analysis of the data was conducted in MATLAB using the script files LDTST5.M and LDTST7.M. Power spectrum estimates were plotted using MATLAB's PSD.M script file from the signal processing toolbox, version. 3.1. The data was windowed using a 4096 point Hanning window thereby producing an equivalent noise bandwidth of 22.75 Hz. The power spectral density plots were plotted in dB's versus Nyquist frequency. Figures 3.10 through 3.13, show demodulator output taken using the program TST5.ASM. Figures 3.14 through 3.17 show output taken using the program TST7.ASM. The estimates of the harmonic distortion were made using the fundamental and first ten harmonics. Both denominator plots in Figure 3.11 and 3.13 show variations due to the non-simultaneous sampling. The denominator term, an estimate which is proportional to the laser power, should be a constant since the simulated interferometric signals have no variation due to laser power fluctuations.

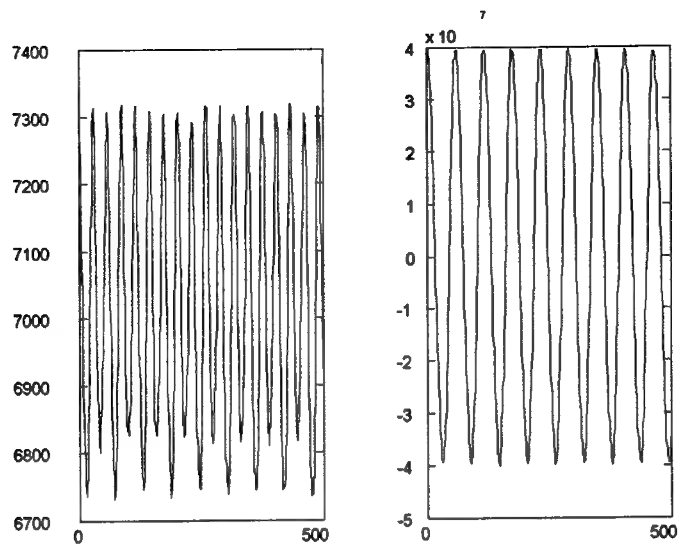

Figure 3.10 Denominator Divided by 2^{16} (Left plot) and Numerator (Right plot) Calculated Using Program TST5.ASM for Input Interferometric Signals of pi Radians Optical Phase Shift at Frequency 0.0176fs.

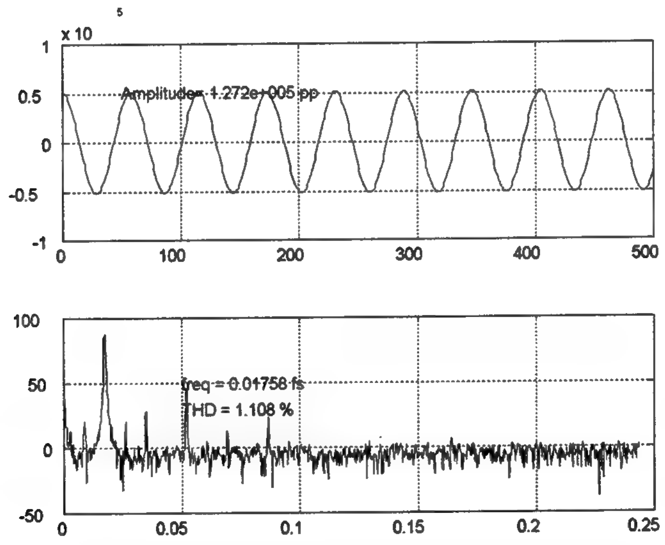

Figure 3.11 Demodulator Output for Interferometric Input of pi Radians Optical Phase Shift at Frequency 0.0176fs. 500 Points of Demodulator Output (Upper plot). Power Spectrum of Output Plotted in dB's vs. Nyquist Frequency (Lower plot).

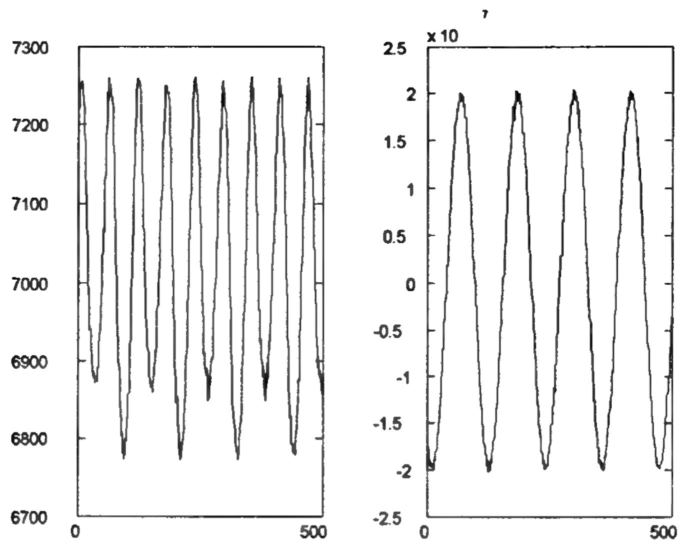

Figure 3.12 Denominator Divided by 2^{16} (Left plot) and Numerator (Right plot) Calculated Using Program TST5.ASM for Input Interferometric Signals of pi Radians Optical Phase Shift at Frequency 0.0088fs.

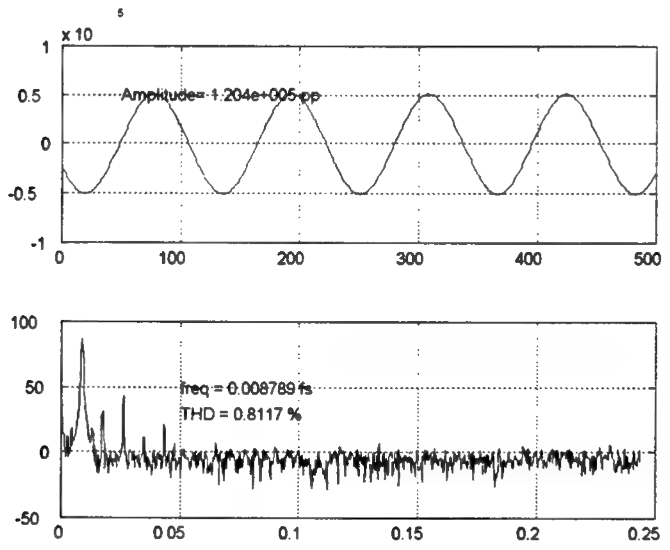

Figure 3.13 Demodulator Output for Interferometric Input of pi Radians Optical Phase Shift at Frequency 0.0088fs. 500 Points of Demodulator Output (Upper plot). Power Spectrum of Output Plotted in dB's vs. Nyquist Frequency (Lower plot).

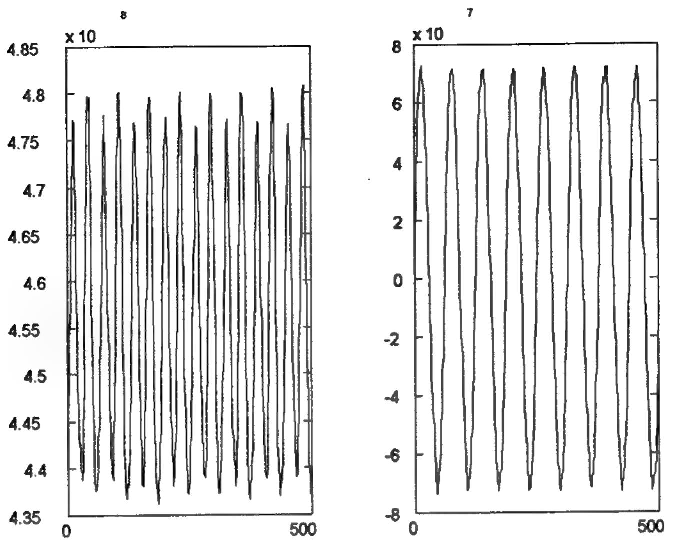

Figure 3.14 Denominator (Left plot) and Numerator (Right plot) Calculated Using Program TST7.ASM for Input Interferometric Signals of pi Radians Optical Phase Shift at Frequency 0.0161fs.

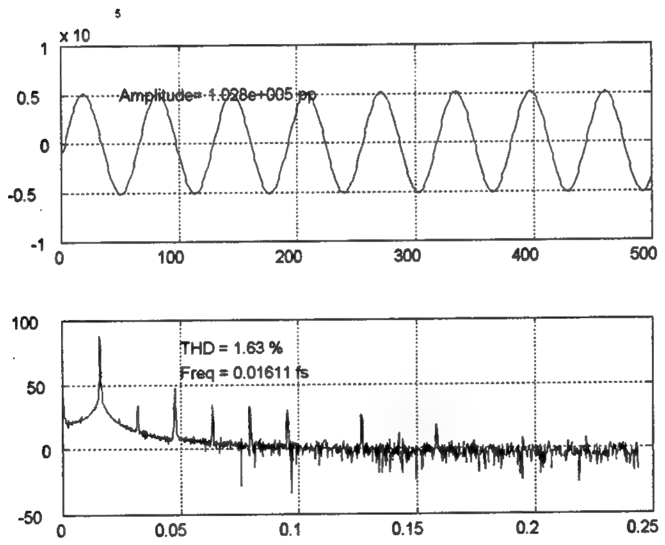

Figure 3.15 Demodulator Output for Interferometric Input of pi Radians Optical Phase Shift at Frequency 0.0161fs. 500 Points of Demodulator Output (Upper plot). Power Spectrum of Output Plotted in dB's vs. Nyquist Frequency (Lower plot).

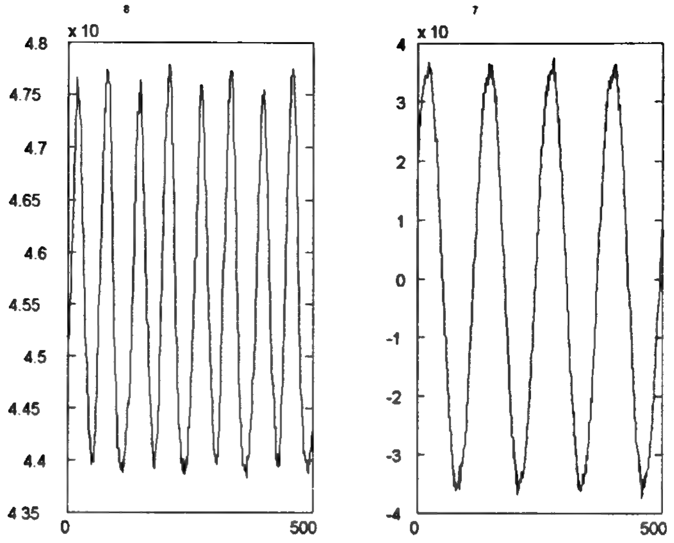

Figure 3.16 Denominator (Left plot) and Numerator (Right plot) Calculated Using Program TST7.ASM for Input Interferometric Signals of pi Radians Optical Phase Shift at Frequency 0.0081fs.

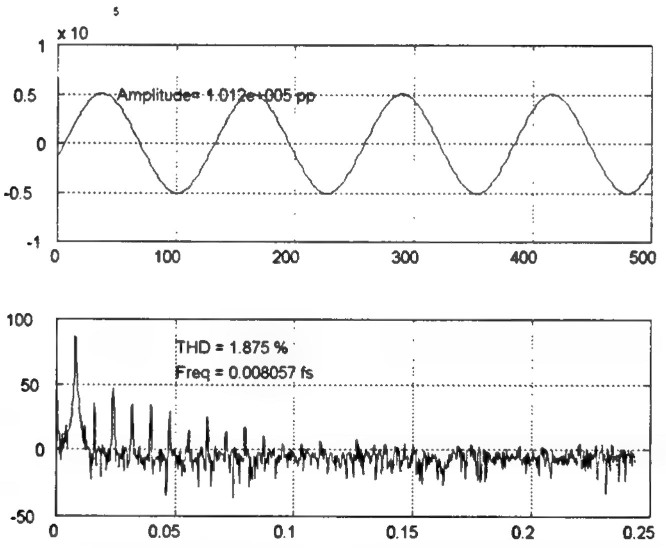

Figure 3.17 Demodulator Output for Interferometric Input of pi Radians Optical Phase Shift at Frequency 0.0081fs. 500 Points of Demodulator Output (Upper plot). Power Spectrum of Output Plotted in dB's vs. Nyquist Frequency (Lower plot).

2. Demodulator Response to Optical Phase Shift Variations.

The demodulator response to variation in optical phase shifts was tested by simulating interferometric signals at a constant frequency of 600 Hz. The optical phase shift produced by the interferometric simulators was adjusted by setting the amplitude of the function generator driving the simulators to three different voltages, 0.01, 0.1 and 0.5 Volts peak. Using the scale factor for the simulators, this was equivalent to 9.33, 93.3, and 466.5 mrad of optical phase shift respectfully. In each run, 8192 points of the denominator, numerator and output signals were calculated by the demodulation programs and later downloaded to the PC. Plotting and analysis of the data was conducted in MATLAB using script files LDTST5.M and LDTST7.M. LDTST7.M is included in Appendix C. Power spectrum estimates were plotted using MATLAB's PSD.M script file from the signal processing toolbox, version. 3.1. The data was windowed using a 4096 point Hanning window thereby producing an equivalent noise bandwidth of 22.75 Hz. The power spectral density plots are plotted in dB's versus the Nyquist frequency. Figures 3.18 through 3.23, show demodulator output taken using the program TST5.ASM. Figures 3.24 through 3.29 show output taken using the program TST7.ASM. The estimates of the harmonic distortion were made using the fundamental and first ten harmonics.

From the plots, it would appear that both demodulation programs can demodulate reasonable ranges of interferometer optical phase shifts. However, demodulation program TST7.ASM, which uses floating point operations appears to have lower denominator fluctuations yet a higher distortion level in the output. Also, demodulation program TST7.ASM executes slightly faster than demodulation program TST5.ASM. Both programs have limited optical phase shift sensitivity because of the non constant denominator signal which is a result of non-simultaneous sampling. I was unable to test the demodulation programs in the multifringe range because of the limitation of the interferometric signal simulators. They could only produce a maximum optical phase shift of 2 pi radians.

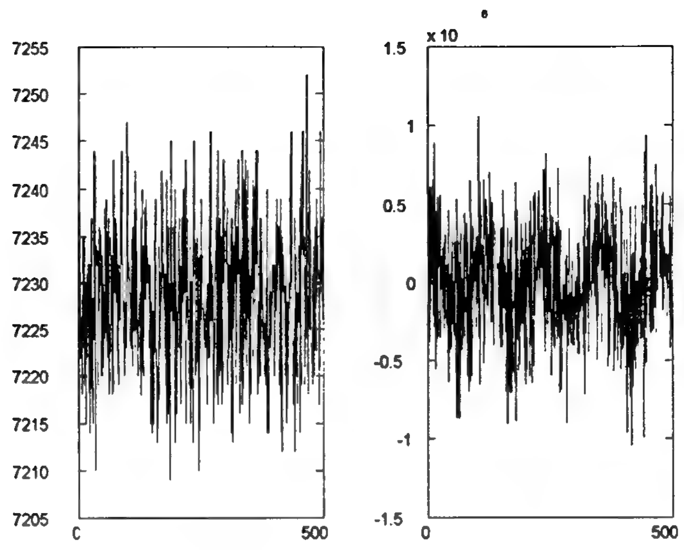

Figure 3.18 Denominator Divided by 2^{16} (Left plot) and Numerator (Right plot) Calculated using Program TST5.ASM for Input Interferometric Signals at 9.3 mrad Optical Phase Shift at Frequency 0.00864fs.

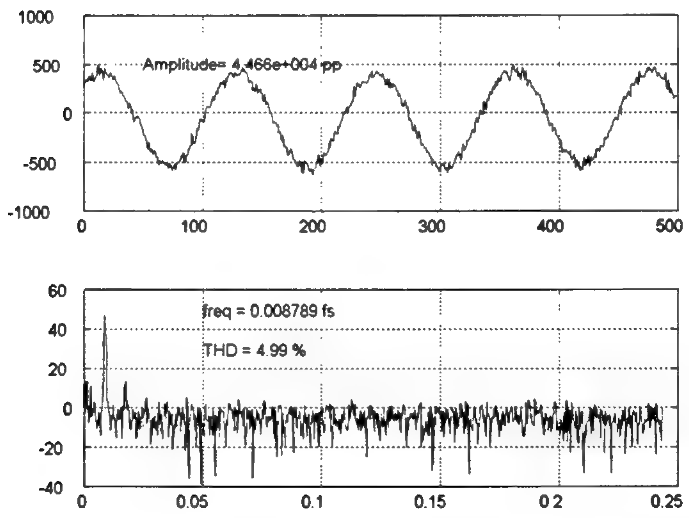

Figure 3.19 Demodulator Output Calculated using Program TST5.ASM for Input Interferometric Signals of 9.3 mrad of Optical Phase Shift at Frequency 0.00864fs. 500 Points of Demodulator Output (Upper plot). Power Spectrum of Output Plotted in dB's vs. Nyquist Frequency (Lower plot).

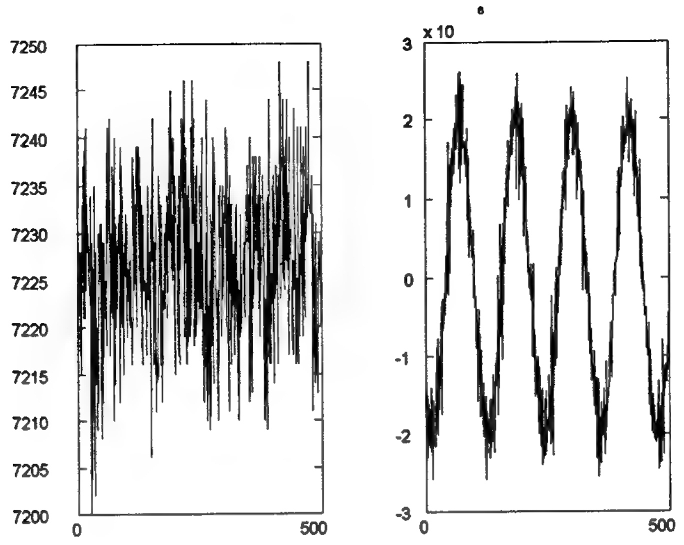

Figure 3.20 Denominator Divided by 2^{16} (Left plot) and Numerator (Right plot) Calculated using Program TST5.ASM for Input Interferometric Signals at 93.3 mrad Optical Phase Shift at Frequency 0.00864fs.

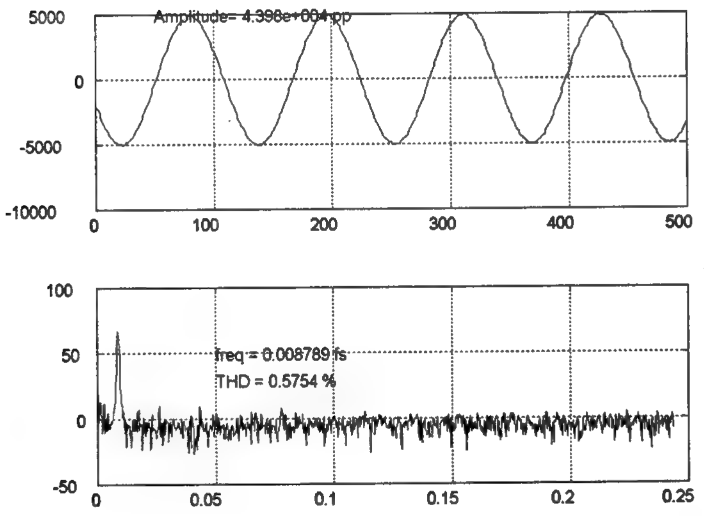

Figure 3.21 Demodulator Output Calculated using Program TST5.ASM for Input Interferometric Signals of 93.3 mrad of Optical Phase Shift at Frequency 0.00864fs. 500 Points of Demodulator Output (Upper plot). Power Spectrum of Output Plotted in dB's vs. Nyquist Frequency (Lower plot).

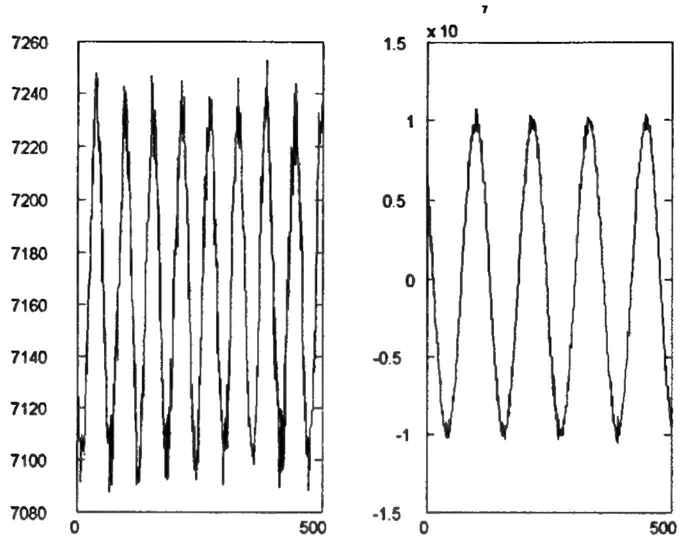

Figure 3.22 Denominator Divided by 2^{16} (Left plot) and Numerator (Right plot) Calculated using Program TST5.ASM for Input Interferometric Signals at 466.5 mrad Optical Phase Shift at Frequency 0.00864fs.

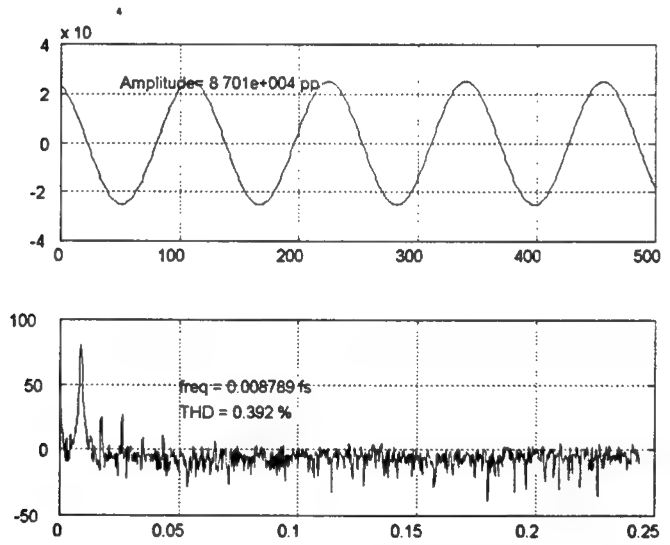

Figure 3.23 Demodulator Output Calculated using Program TST5.ASM for Input Interferometric Signals of 466.5 mrad of Optical Phase Shift at Frequency 0.00864fs. 500 Points of Demodulator Output (Upper plot). Power Spectrum of Output Plotted in dB's vs. Nyquist Frequency (Lower plot).

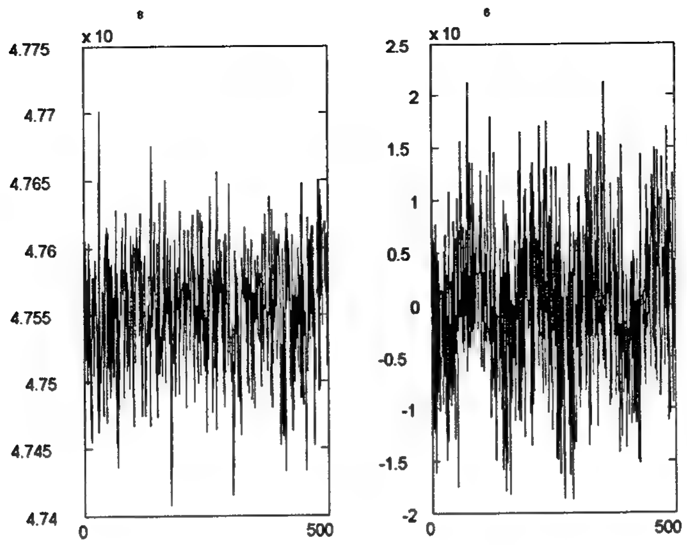

Figure 3.24 Denominator (Left plot) and Numerator (Right plot) Calculated using Program TST7.ASM for Input Interferometric Signals at 9.33 mrad Optical Phase Shift at Frequency 0.00792fs.

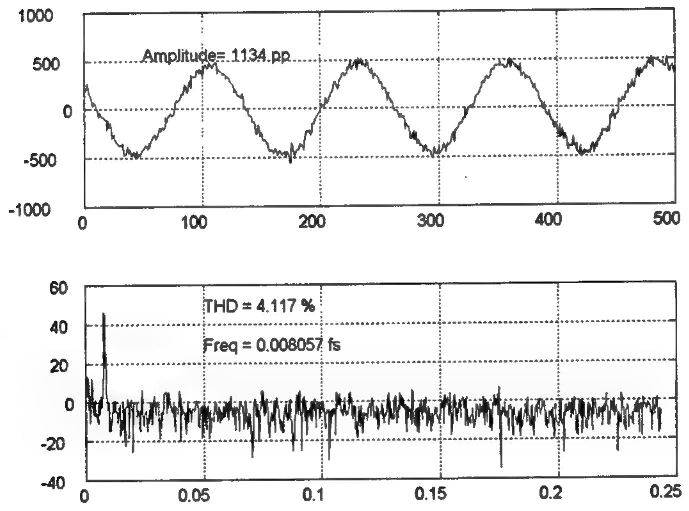

Figure 3.25 Demodulator Output Calculated using Program TST7.ASM for Input Interferometric Signals of 9.33 mrad of Optical Phase Shift at Frequency 0.00792fs. 500 Points of Demodulator Output (Upper plot). Power Spectrum of Output Plotted in dB's vs. Nyquist Frequency (Lower plot).

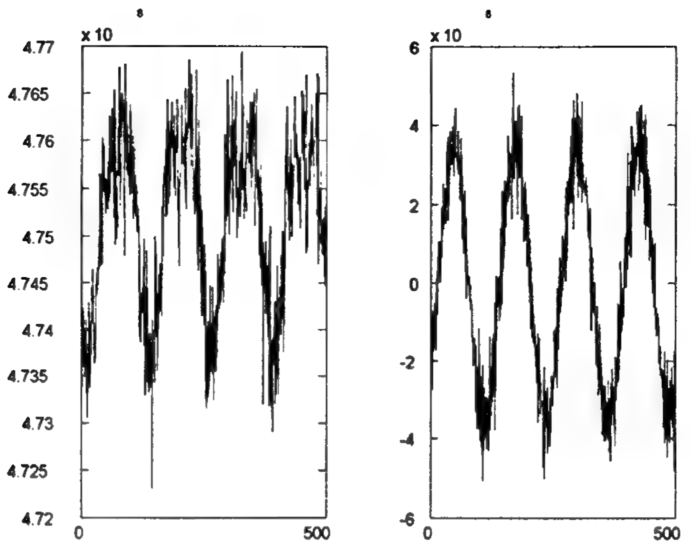

Figure 3.26 Denominator (Left plot) and Numerator (Right plot) Calculated using Program TST7.ASM for Input Interferometric Signals at 93.3 mrad Optical Phase Shift at Frequency 0.00792fs.

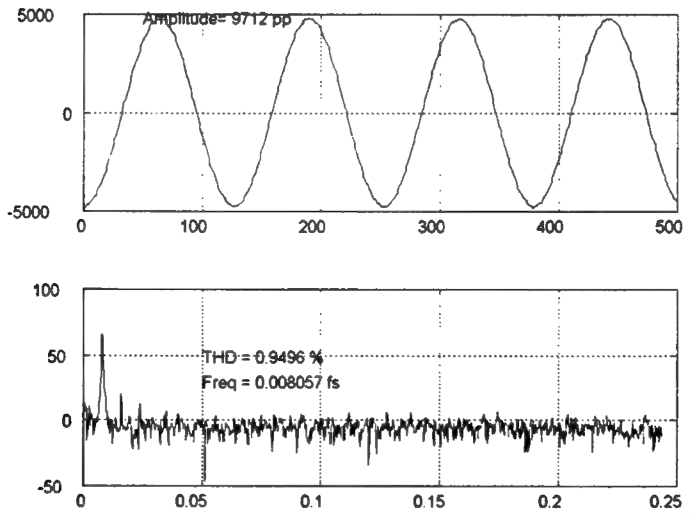

Figure 3.27 Demodulator Output Calculated using Program TST7.ASM for Input Interferometric Signals of 93.3 mrad of Optical Phase Shift at Frequency 0.00792fs. 500 Points of Demodulator Output (Upper plot). Power Spectrum of Output Plotted in dB's vs. Nyquist Frequency (Lower plot).

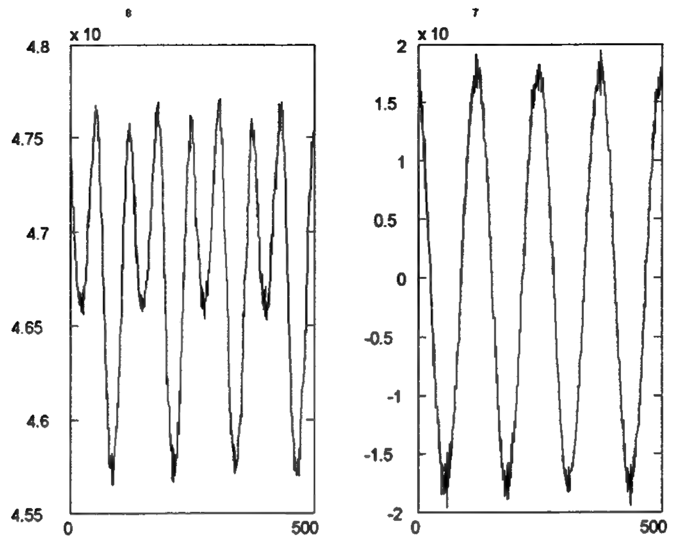

Figure 3.28 Denominator (Left plot) and Numerator (Right plot) Calculated using Program TST7.ASM for Input Interferometric Signals at 466.5 mrad Optical Phase Shift at Frequency 0.00792fs.

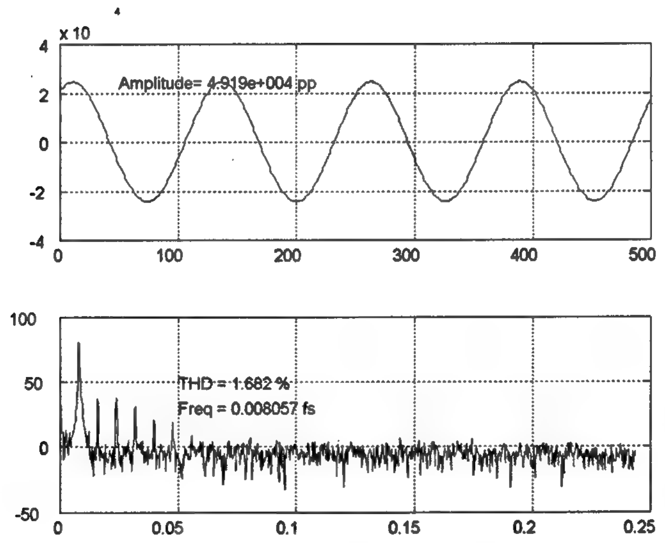

Figure 3.29 Demodulator Output Calculated using Program TST7.ASM for Input Interferometric Signals of 466.5 mrad of Optical Phase Shift at Frequency 0.00792fs. 500 Points of Demodulator Output (Upper plot). Power Spectrum of Output Plotted in dB's vs. Nyquist Frequency (Lower plot).

IV. DESIGN OF AN EXTERNAL SAMPLE AND HOLD CIRCUIT

This chapter discusses the design of an external sample and hold circuit which provided four channels of simultaneously sampled inputs. The PC31 limits analog input to two channels which can be sampled simultaneously using one DSP102. To obtain three analog input channels required the use of the internal multiplexing which resulted in non-simultaneous sampling. The results from the demodulation using non-simultaneous sampling clearly show an increase in output distortion caused by a non-stable denominator. Therefore, it was decided to build an external circuit which would provide simultaneous sampling.

A. DESIGN

The heart of the circuit involves three SHC5320 sample and hold IC's and an HI 507a multiplexer, refer to Figure 4.1, both from Burr-Brown [Ref 18]. The sample and hold IC's were conFigured in the non-inverting unity gain mode. They were supplied with ±15 volts and the inputs were protected with input limiting diodes. Input to the three sample and holds can be from either the interferometric simulators or photo detected 3x3 coupler outputs. The outputs from the sample and hold are sent to input pair 1A and 1B, and input 2A of the multiplexer. Input 2B of the multiplexer is grounded. The outputs, OUT A and OUT B from the multiplexer are connected to inputs A and B of the DSP102.

The timing for the multiplexer and sample and hold circuitry is generated by the PC31 using internal timer #1, and the 82C54 timer, and by the external board using an 74HCT73, 74HCT109, and a 74HCT04. The timing of the sample and hold board is show in Figure 4.2 for the designed sampling frequency of 100 kHz. The PC31's internal timer #1 (TCLK1) is conFigured to run as a square wave clock source with a frequency dependent upon the value loaded into the period register. A value of decimal 41 loaded into this register generated a clock frequency of 203.252 kHz. TCLK1 is used to generate the master timing clock. TCLK1 is routed from the PC31 to the external board where it is divided by two using one half of a 74HCT73 JK flip-flop, and one half of a 74HCT109 JK flip-flop. The 74HCT73 is rising edge triggered while the 109 is falling edge triggered. These two chips generate two timing clocks at one half the frequency of TCLK1, 101.626

kHz, one in phase with the rising edge of TCLK1, the other in phase with falling edge of TCLK1.

The rising edge of TCLK1 is used to gate timer/counter 0 of the 82C54. The 82C54 is a CMOS programmable interval timer. There is one such timer on the PC31 DSP board. The 82C54 has three independent 16 bit counters, which can operate in either BCD or binary counting. Each counter has six programmable modes of operation. Timer 0 is conFigured for mode 1, hardware retriggerable one-shot operation. When triggered by the rising edge of TCLK1, the output of timer 1, OUT0, goes low on the clock pulse following the trigger and remains low for the number of clock cycles loaded into the count register. The input clock source for counter 0 is the peripheral clock driver set to one quarter of the system clock, 8.33 MHz. Counter 0 is loaded with a count of decimal 9 which produces an output low signal of about 1.16 μsec. OUT0 is then inverted by one 6th of a 74HCT04 and used as the convert signal for the DSP102.

Timer 1 of the 82C54 is also conFigured in mode 1, hardware retriggerable one-shot. The output from the 74HCT109, is used to trigger timer 1 on input GATE1 to the 82C54. OUT1 goes low on the clock pulse following the trigger and remains low for the number of clock cycles loaded in its count register. A value of decimal 58 loaded into the count register produces a low pulse of 7.2 μsec on OUT1. This signal is also inverted by one sixth of a 74HCT04 and used to switch the Hold/Sample* input to the SHC5320's. The output from the 74HCT73 is used to drive pin A0 of the multiplexer. This pin when brought low selects input pair #1. A high on pin A0 selects input pair #2.

The SHC5320's are brought to the hold state within 200 nsec of the rising edge of TCLK1. Then the DSP102 is triggered by the falling edge of OUT0*, 1.16 μsec after the rising edge of TCLK1. This gives the held output from the SHC5320's approximately 1 μ sec to settle before being sampled. At 2.5μsec after the rising edge of TCLK1 the falling edge of TCLK1 switches the multiplexer from input pair 1, (interferometric signals a and c), to input pair 2, (interferometric signal b and ground). This gives the multiplexer outputs approximately 3.6 μsec in which to settle before being sampled, more than sufficient time to achieve 0.01% accuracy. 4.8 μsec after the convert command to the DSP102 , the first conversion is complete and interrupts the processor. The next conversion is signaled at 5.0 μsec from the first, and after 1 μsec the SHC5320's are brought to the sample state. This gives the sample and holds about 1.4 μsec in which to re-track the input signal.

B. RESULTS OF CIRCUIT TESTING

With the external sample and hold circuit connected to the PC31, I was able to obtain four channels of analog input which could be simultaneously sampled at a maximum of 100 kHz. However, because the circuit was built on an external Proto-Board, with little consideration of interference, the noise levels at the input were greatly elevated, roughly on the order of 0.5 mV per channel. Nonetheless, I was able to implement the algorithm using this sampling circuit and the denominator signal was relatively constant, and the output showed signs of reduced distortion. A permanent board is currently being constructed by Professor Keolian for use with the PC31. This circuit was engineered to reduce the unwanted effects of external noise sources.

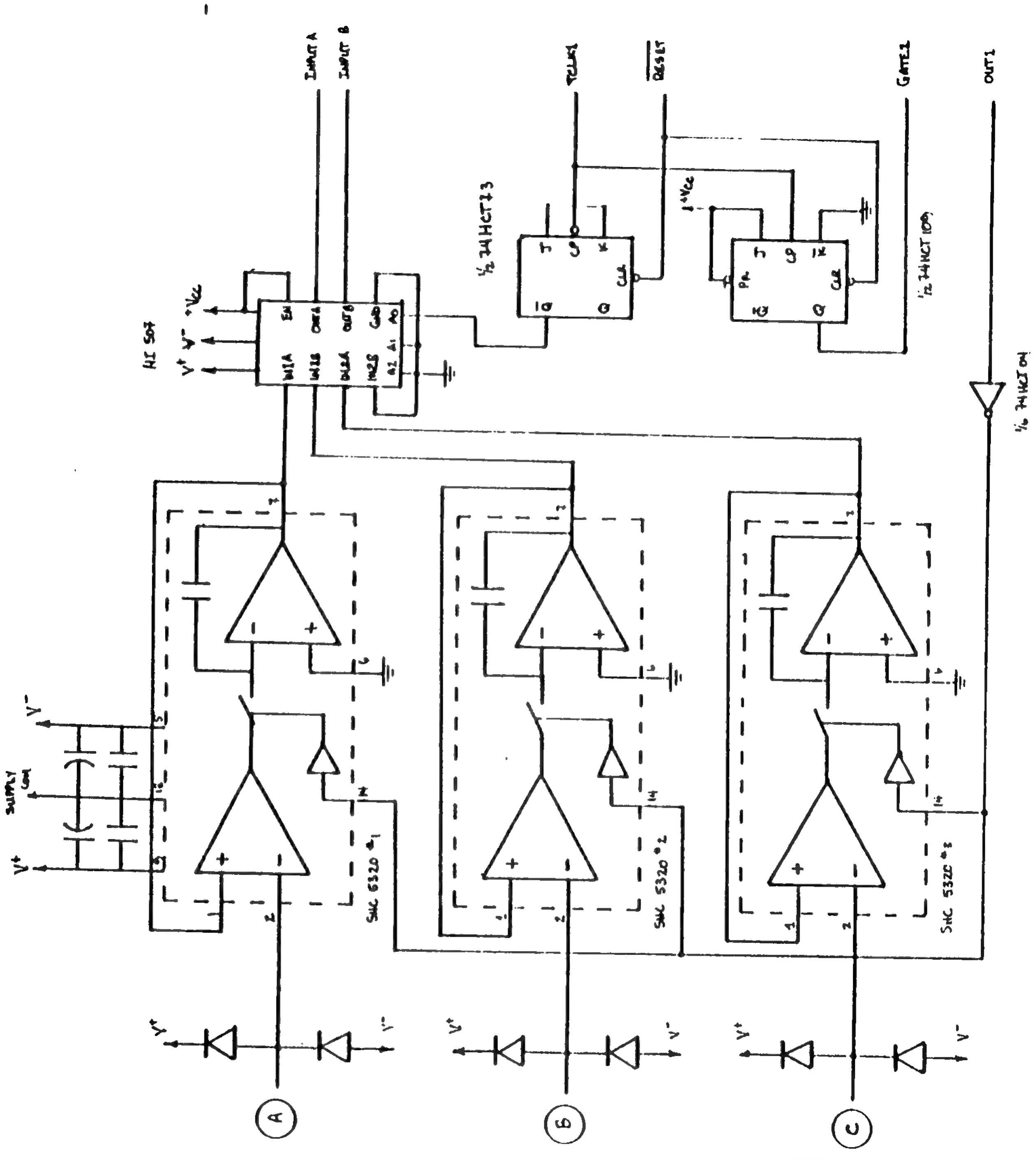

Figure 4.1 Schematic of external sample and hold circuitry.

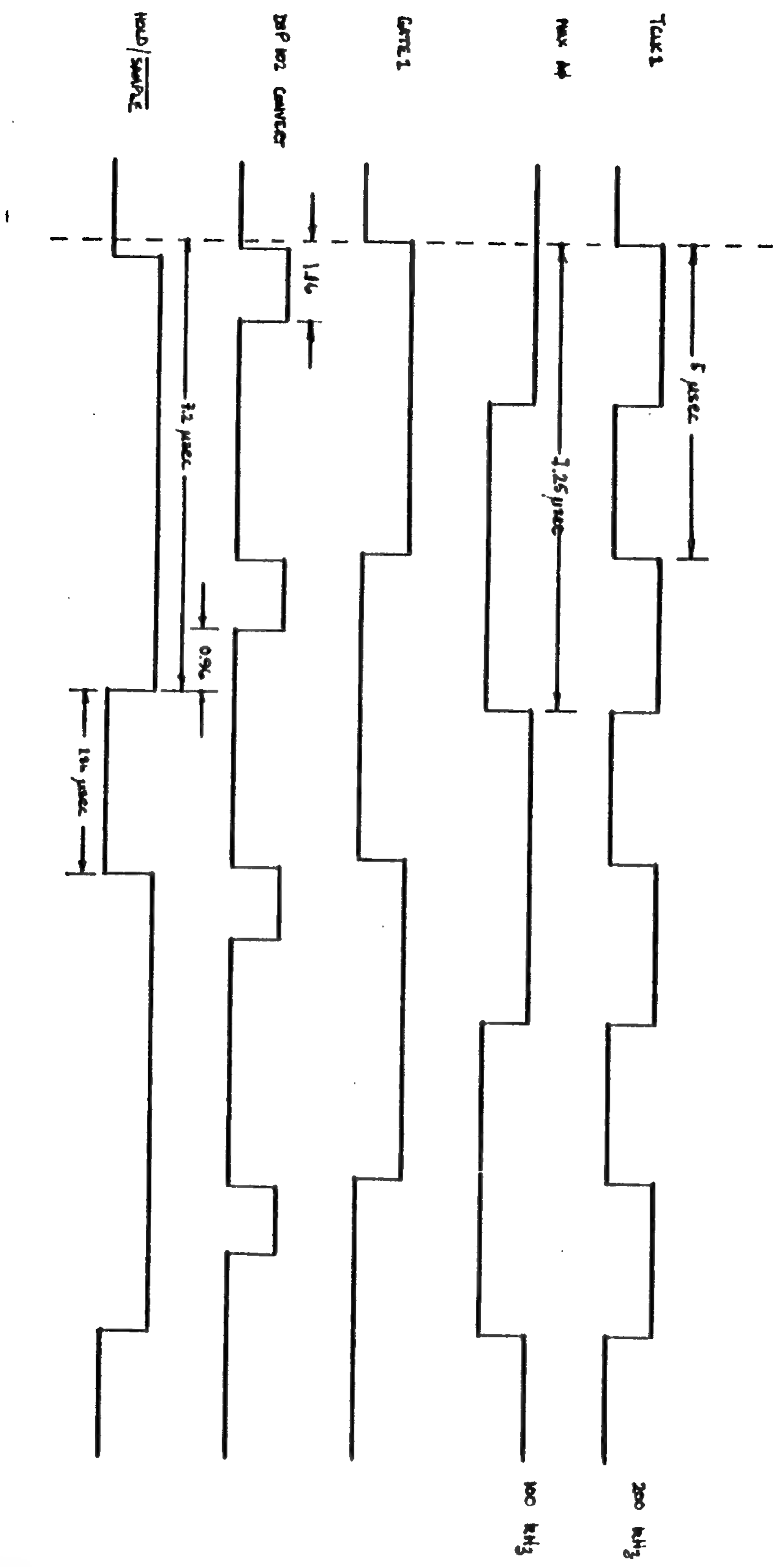

Figure 4.2 Timing diagram for external sample and hold circuit.

V. CONCLUSIONS AND RECOMMENDATIONS

This thesis has demonstrated that implementation of the symmetric demodulation scheme by digital signal processing hardware is possible and in fact extremely practical. It was shown that by using a DSP plug-in board, based upon the TMS320C31 DSP processor, the demodulation of three interferometric signals could execute in as little time as 10 μsec. Using a first difference approach was suitable for calculation of the signal derivatives. The algorithm can be implemented using either fixed point integer or floating point operations, although the fixed point algorithm may take longer to execute. The demodulation execution time fixes the maximum sampling rate of the demodulator and thus the maximum allowable fringe rate, which is directly proportional to the sampling rate. The sensitivity of the demodulator is fixed by the size of the analog-to-digital quantization which is a function of the ADC. For a give sampling rate set by the demodulation process, increasing the conversion word size of the analog-to-digital converters will increase the sensitivity. Use of delta sigma converting ADCs, as suggested by McGinnis [Ref 9], was never considered, however, their usefulness could easily be verified in the future.

It was also shown that non-simultaneous sampling of the input interferometric signals leads to elevated levels of distortion in the output. These effects were not quantified and are left for follow-on research. An external sample and hold circuit was built and tested which would allow for the simultaneous sampling of up to four analog channels at a maximum of 100 kHz. This was shown to produce a fairly stable denominator signal which lead to reduced distortion in the output.

APPENDIX A. DSP CARD: PC31 BY INNOVATION INTEGRATION

The PC31 is a very high performance, IBM PC plug-in co-processor featuring the Texas Instruments TMS320C31 DSP processor. Lt. B. McGinnis had previously surveyed many DSP boards, and the PC31 was selected for its processing capabilities, analog section, good input-output support, and low cost. The TMS320C31 DSP processor meets all of the requirements for high-speed, real time applications: 17 MIPS/ 33 MFLOPS of processing power, hardware floating point multiplier/accumulator, DMA controller, on chip synchronous serial port ADC and DAC.

Software is interactively developed and tested using the optimizing ANSI compatible C compiler. Applications are developed and tested using the integer and floating point math, analog/digital I/O and video display libraries provided with the Developers package. All systems and application procedures may conveniently be interactively edited, tested and executed at full machine speed.

The Board serves as its own development system using the system monitor resident in the IBM PC host in combination with other essential software tools supplied with the Developers package. Once an application has been created, it may be embedded in PROM for stand alone operation or downloaded to the board from within the system monitor. The PC31 communicates with an IBM PC/AT, or equivalent, via a high speed dual port memory arrangement which allows access to all PC31 memory and peripherals from the IBM PC.

Interfacing with external peripherals is accomplished with two 10-pin serial ports, one 37-pin analog connector, one 50-pin digital connector or one 108 pin expansion connector. Additionally, the PC31 is compatible with the full line of DSP-LINK analog and digital I/O cards.

APPENDIX B. ANALOG INTERFEROMETRIC SIGNAL SIMULATOR

The analog interferometric signal simulator built by Dr. David Gardner, is similar to that described in [Ref 7:Appendix C]. Additional modifications were made to the circuit and are described in [Ref 19:pp 73-75]. The simulator was built around Analog Device's AD639 Universal Trigonometric Function Converter (refer to Figure B1). Adjustments on the simulator provide for the setting of the static phase angle and output signal amplitude, however, the drift angle term is not time-varying as it would be in an actual system. By configuring three simulators together and adjusting the static phase angle of each, three interferometric waveforms separated by 120° can be generated for any bias angle.

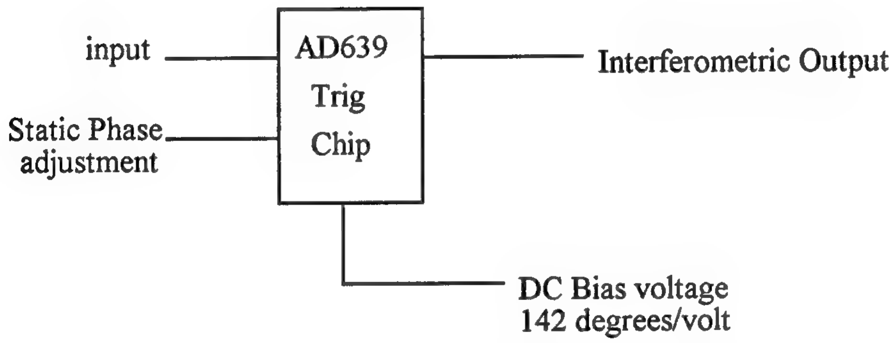

Figure B1 Block diagram of interferometric signal simulator taken from Ref 9

The Analog Devices AD639 Universal Trigonometric Function Converter is conFigured to produce the sine of its input. A one volt input is equivalent to an input argument of 50°. The gain of the circuit built by Gardner [Ref 7] was set to 2.85. Thus the static phase adjustment input equates to,

$$2.85 \times 50°/\mathrm{Volt} \;=\; 142°/\mathrm{Volt}, \tag{B.1}$$

so that, with one simulator adjusted to zero volts static phase output and the other two static phase outputs adjusted to ±0.845 V, three interferometric signals symmetrically phase shifted 120° relative to each other will be produced.

As designed, the simulators produced a maximum output of 3.6 V peak to peak. This was well within the range of allowed for input voltages to the DSP102. However, they are not able to produce the dc offset voltage which is associated with real interferometric signals. The simulators were able to generate outputs for input signal amplitudes of up to 8.0 Volts peak-to-peak ($\cong$ 2 pi radians). Above 8.0 volts, the simulators become severely distorted due to saturation in the analog electronics. The average scale factor for the three interferometric simulators was reported to be 0.933 rad/Volt [Ref 9 p 51]. The average phase noise was reported to be 0.48 ± 0.005 μrad/√ Hz measured at 1 kHz in a one Hz bandwidth [Ref 19:p 64].

APPENDIX C. MISCELLANEOUS

This appendix contains the following:

1. Turbo C programs written for this thesis.

    TCDEMOD.C

2. MATLAB script files written for this thesis.

    PLOTDAT.M

    DEMOD.M

    PLOTABC.M

    LDTST3.M

    LDTST7.M

3. PC31 C programs written for this thesis.

    TMSDEMOD.C

4. PC31 Assembly language programs written for this thesis.

    TST3.ASM

    TST5.ASM

    TST7.ASM

A. PROGRAM TCDEMOD.C

```
/*********************************************************************
*
*          TCDEMOD.C - Test routine written in Turbo C.
*           This program generates the three channels output from a 3x3
*           fiber optic interferometer.  The 3 channels are then
*           demodulated using a symmetric demodulation scheme using
*           integer arithmetic.  The output is accumulated in the variable
*           phase
*
*           Written by : LCdr Brenner 09 Feb 1994
*
*           last modified : 11 Aug 94
*
*********************************************************************/

#include "stdio.h"
#include "math.h"
#define num_samples 1024
#define onethird 1.0/3.0
#define decay 0.999

void main(void);
void gen_signal(void);

void main()
{
          extern int A[num_samples];
          extern int B[num_samples];
          extern int C[num_samples];
          long int output[num_samples];
          long int denom[num_samples];
          long int numer[num_samples];
          long int a[num_samples];
          long int b[num_samples];
          long int c[num_samples];
          long int adot[num_samples];
          long int bdot[num_samples];
          long int cdot[num_samples];
          long int a_temp, b_temp, c_temp, numer1, numer2, numer3;
          int i, offset;
          FILE *fp;

          a_temp = 0; b_temp =0; c_temp = 0;
          numer1 = 0; numer2 = 0; numer3 = 0;
          offset = 0;

          fp = fopen("demod.dat", "w+b");
```

```
        gen_signal();

        for(i = 0; i < num_samples; i++)
        {
                offset = (int)(onethird*(A[i] + B[i] + C[i]));
                a[i] = (A[i] - offset);
                b[i] = (B[i] - offset);
                c[i] = (C[i] - offset);

                denom[i] = (a[i]*a[i] + b[i]*b[i] + c[i]*c[i]) >> 16;
                adot[i] = a[i] - a_temp;
                bdot[i] = b[i] - b_temp;
                cdot[i] = c[i] - c_temp;
                numer1 = a[i]*(cdot[i] - bdot[i]);
                numer2 = b[i]*(adot[i] - cdot[i]);
                numer3 = c[i]*(bdot[i] - adot[i]);
                numer[i] = numer1 + numer2 + numer3;

                output[i] = (long int)(decay*output[i-1]) + numer[i]/denom[i];

                a_temp = a[i]; b_temp = b[i]; c_temp = c[i];
        }
        fwrite(a, sizeof(long int), num_samples, fp);
        fwrite(adot, sizeof(long int), num_samples, fp);
        fwrite(b, sizeof(long int), num_samples, fp);
        fwrite(bdot, sizeof(long int), num_samples, fp);
        fwrite(c, sizeof(long int), num_samples, fp);
        fwrite(cdot, sizeof(long int), num_samples, fp);
        fwrite(denom, sizeof(long int), num_samples, fp);
        fwrite(numer, sizeof(long int), num_samples, fp);
        fwrite(output, sizeof(long int), num_samples, fp);
        fclose(fp);
}

/*
*
* Routine to generate interferemetric waveforms
*
*/
int A[num_samples];
int B[num_samples];
int C[num_samples];

void gen_signal()
{
        int i, scale;
        float Ac, f, fs, Light_amp, pi;
        float theta[num_samples];

        pi = 4.0*atan(1.0);
        fs = 1.0/128.0;
        Ac = pi;
        Light_amp = 0.0;
        scale = 32000;

        for(i = 0; i < num_samples; i++)
        {
                theta[i] = 2.0*pi*fs*i;
                A[i] = (Light_amp + cos(Ac*sin(theta[i]) - pi/6))*scale;
                B[i] = (Light_amp + cos(Ac*sin(theta[i]) + pi/2))*scale;
                C[i] = (Light_amp + cos(Ac*sin(theta[i]) + 7*pi/6))*scale;
        }
}
```

B. PROGAM PLOTDAT.M

```
%************************************************************
%
% PLOTDAT.M - Plots data generated by the C program DEMOD1.C
%
%   Written by: LCdr Brenner    10 Aug 94
%
%************************************************************

clear all, clc, clf

N = 2048;

A = ld16bit('c:\tc\thesis\demod1.dat',N,0);
B = ld16bit('c:\tc\thesis\demod1.dat',N,N);
C = ld16bit('c:\tc\thesis\demod1.dat',N,2*N);
output = ld32bit('c:\tc\thesis\demod1.dat',N,3*N/2);

L = 200;
l = 0:L-1;
figure(1)
plot(l,A(1:L),':',l,B(1:L),'--',l,C(1:L))

figure(2)
subplot(211),plot([0:1999],output(1:2000))
disp(['Max output = ',num2str(max(output)),' Min output = ',num2str(min(output))])

Nfft = N; fs = 100e3;
w = hanning(Nfft);
[Pxx,fx] = psd(output,Nfft,fs,w,256);
Pxx = Pxx*norm(w)^2/sum(w)^2;
subplot(212),plot(fx/fs,10*log10(Pxx)),grid
%[x,y] = ginput(6);
%y = 10.^(y/20);
%THD = sum(y(2:6))/y(1)*100;
%text(10e3,110,['THD = ',num2str(THD),' %'])
```

C. PROGRAM DEMOD.M

```
%**********************************************************************
%
% DEMOD.M - this program plots theoretical interferemetric signals
%       for a three leg interferometer.  And then demodulates them using
%       a symetric demodulation technique.
%
%       written by: LCdr Brenner  7 June 94
%       Last modified : 20 July 94
%
%**********************************************************************

clear all,clf,clc

N = 1024;
%diary demod1.txt

%set acoustic amplitude plus noise.
noise = 0;
phase_shift = pi;
Ac = phase_shift*ones(1,N) + noise*randn(1,N);
disp(['optical phase shift = ',num2str(phase_shift),' radians'])

%set sampling frequency f/fs
fs = 100e3; f = 600;
disp(['sampling freq = ',num2str(fs),' Hz'])
disp(['acoustic frequency = ',num2str(f),' Hz'])

%calculate sampled phase angle
theta = (2*pi*f/fs)*[1:N];

%set phase noise term (static phase difference plus drift)
phid_noise = 0.0;
phid = phid_noise*rand(1,N);

%calculate input signal
signal = Ac.*sin(theta + phid);

%set laser amplitude and measurement noise
Dc = 0.5/2.75;
samp = 5.0;
sampling_noise = samp.*randn(1,N);

%generate theoretical interferemetric signals
scale = 2^15;
A = fix((Dc + cos(signal - pi/6))*scale) + sampling_noise;
B = fix((Dc + cos(signal + pi/2))*scale) + sampling_noise;
C = fix((Dc + cos(signal + 7*pi/6))*scale) + sampling_noise;
```

```
% intialize variables
decay = 0.99;
offset1 = fix(1/3*(A(1) + B(1) + C(1)));
offset2 = fix(1/3*(A(2) + B(2) + C(2)));
a1 = fix(A(1) - offset1); a2 = fix(A(2) - offset2);
b1 = fix(B(1) - offset1); b2 = fix(B(2) - offset2);
c1 = fix(C(1) - offset1); c2 = fix(C(2) - offset2);
denom = fix((a1^2 + b1^2 + c1^2)/2^16);
adot = fix(a2 - a1); atemp = a1;
bdot = fix(b2 - b1); btemp = b1;
cdot = fix(c2 - c1); ctemp = c1;
numer = fix(a1*(cdot - bdot) + b1*(adot - cdot) + c1*(bdot - adot));
last_output = numer/denom;

%run algorithm
for k = 1:N
        offset = fix(1/3*(A(k) + B(k) + C(k)));
        a = A(k) - offset;
        b = B(k) - offset;              %subtract dc from interferometric signals
        c = C(k) - offset;
        denom(k) = fix((a^2 + b^2 + c^2)/2^16);     %calculate factor that accounts for laser power
        adot(k) = a - atemp;
        bdot(k) = b - btemp;          %calcluate derivatives of signals
        cdot(k) = c - ctemp;
        numer1(k) = fix(a*(cdot(k) - bdot(k)));
        numer2(k) = fix(b*(adot(k) - cdot(k)));
        numer3(k) = fix(c*(bdot(k) - adot(k)));
        numer(k) = fix(numer1(k) + numer2(k) + numer3(k));
        output(k) = fix(decay*last_output + numer(k)/denom(k));

        atemp = a; btemp = b; ctemp = c; last_output = output(k);
end

figure(1)
L = 200; l = 0:L-1;
subplot(131),plot(l,A(1:L))
subplot(132),plot(l,B(1:L))
subplot(133),plot(l,C(1:L))
disp(['max A = ',num2str(max(A)),' min A = ',num2str(min(A))])
disp(['max B = ',num2str(max(B)),' min B = ',num2str(min(B))])
disp(['max C = ',num2str(max(C)),' min C = ',num2str(min(C))])

figure(2)

subplot(311),plot(adot)
disp(['max adot = ',num2str(max(adot)),' min adot = ',num2str(min(adot))])
subplot(312),plot(bdot)
disp(['max bdot = ',num2str(max(bdot)),' min bdot = ',num2str(min(bdot))])
subplot(313),plot(cdot)
disp(['max cdot = ',num2str(max(cdot)),' min cdot = ',num2str(min(cdot))])
```

```
figure(3)

subplot(311),plot(numer1)
disp(['max numer1 = ',num2str(max(numer1)),' min numer1 = ',num2str(min(numer1))])
subplot(312),plot(numer2)
disp(['max numer2 = ',num2str(max(numer2)),' min numer2 = ',num2str(min(numer2))])
subplot(313),plot(numer3)
disp(['max numer3 = ',num2str(max(numer3)),' min numer3 = ',num2str(min(numer3))])

figure(4)

subplot(211),plot(numer(1:300))
disp(['max numer = ',num2str(max(numer)),' min numer= ',num2str(min(numer))])
subplot(212),plot(denom(1:300))
disp(['max denom = ',num2str(max(denom)),' min denom = ',num2str(min(denom))])

figure(5)
subplot(211),plot(output(1:1000))

Nfft = 1024;
w = hanning(Nfft);
[Pxx,fx] = psd(output,Nfft,fs,w,256);
Pxx = Pxx*norm(w)^2/sum(w)^2;
subplot(212),plot(fx/fs,10*log10(Pxx)),grid
[x,y] = ginput(11);
y = 10.^(y/20);
THD = sum(y(2:11))/y(1)*100;
text(0.1,100,['THD = ',num2str(THD),' %'])

disp('-------------------------------------------------------------------')
diary off
```

D. PROGRAM PLOTABC.M

```
%****************************************************
%
% PLOTABC.M - Plots test data taken from the PC31.
%       This script file assumes that data has been
%       taken from the PC31 using TST3.ASM
%
%       Written by LCdr Brenner 10 Aug 94
%
%****************************************************

clear all, clf, clc
Nfft = 4096;
fs = 75757.6;
w = hanning(Nfft);
%a0 = 0.35875; a1 = 0.48829; a2 = 0.14128; a3 = 0.01168; n = 0:Nfft-1;
%w = a0 - a1*cos(2*pi*n/Nfft) + a2*cos(4*pi*n/Nfft) - a3*cos(6*pi*n/Nfft);

a_gnd = ld16bit('a_gnd.dat');
b_gnd = ld16bit('b_gnd.dat');
c_gnd = ld16bit('c_gnd.dat');

afull = ld16bit('amux600.dat');
bfull = ld16bit('bmux600.dat');
cfull = ld16bit('cmux600.dat');
figure(1)
subplot(211),plot(afull(1:1000))
%title(['Data A multiplexed on channel a (sampling freq = ',num2str(fs),' Hz)'])
text(100,3.5e4,['Max = ',num2str(max(afull)),'Min = ',num2str(min(afull))])
[Paa,fa] = psd(afull(1:Nfft),Nfft,fs,w,256);
Paa = Paa*norm(w)^2/sum(w)^2;
subplot(212),plot(fa(1:1000)/fs,10*log10(Paa(1:1000)))
%ylabel('Magnitude - dB')
SNR = 10*log10(max(Paa)/mean(Paa(1000:Nfft/2)));
text(0.025,80,['SNR = ',num2str(SNR),' dB'])
[x,y] = ginput(11);
y = 10.^(y/20);
THD = sum(y(2:11))/y(1)*100;
text(0.025,50,['THD = ',num2str(THD),' %'])
figure(2)
subplot(211),plot(bfull(1:1000))
%title(['Data B multiplexed on channel a (sampling freq = ',num2str(fs),' Hz)'])
text(100,3.5e4,['Max = ',num2str(max(bfull)),'Min = ',num2str(min(bfull))])
[Pbb,fb] = psd(bfull(1:Nfft),Nfft,fs,w,256);
Pbb = Pbb*norm(w)^2/sum(w)^2;
subplot(212),plot(fb(1:1000)/fs,10*log10(Pbb(1:1000)))
%ylabel('Magnitude - dB')
SNR = 10*log10(max(Paa)/mean(Paa(1000:Nfft/2)));
text(0.025,80,['SNR = ',num2str(SNR),' dB'])
```

```
[x,y] = ginput(11);
y = 10.^(y/20);
THD = sum(y(2:11))/y(1)*100;
text(0.025,50,['THD = ',num2str(THD),' %'])
figure(3)
subplot(211),plot(cfull(1:1000))
%title(['Data C on channel b  (sampling freq = ',num2str(fs),' Hz)'])
text(100,3.5e4,['Max = ',num2str(max(cfull)),'Min = ',num2str(min(cfull))])
[Pcc,fc] = psd(cfull(1:Nfft),Nfft,fs,w,256);
Pcc = Pcc*norm(w)^2/sum(w)^2;
subplot(212),plot(fc(1:1000)/fs,10*log10(Pcc(1:1000)))
%ylabel('Magnitude - dB')
SNR = 10*log10(max(Paa)/mean(Paa(1000:Nfft/2)));
text(0.025,80,['SNR = ',num2str(SNR),' dB'])
[x,y] = ginput(11);
y = 10.^(y/20);
THD = sum(y(2:11))/y(1)*100;
text(0.025,50,['THD = ',num2str(THD),' %'])
figure(4)
subplot(211),plot(a_gnd(1:500))
%title(['Data A multiplexed on channel a (sampling freq = ',num2str(fs),' Hz)'])
text(50,15,['Max = ',num2str(max(a_gnd)),' Min = ',num2str(min(a_gnd))])
text(50,-15,['Mean = ',num2str(mean(a_gnd))])
[Paa,fa] = psd(a_gnd(1:Nfft),Nfft,fs,w,256);
Paa = Paa*norm(w)^2/sum(w)^2;
subplot(212),plot(fa/fs,10*log10(Paa))
%ylabel('Magnitude - dB')
text(5e3,-10,['Mean = ',num2str(10*log10(mean(Paa))),' dB'])
figure(5)
subplot(211),plot(b_gnd(1:500))
%title(['Data B multiplexed on channel a (sampling freq = ',num2str(fs),' Hz)'])
text(50,15,['Max = ',num2str(max(b_gnd)),' Min = ',num2str(min(b_gnd))])
text(50,-15,['Mean = ',num2str(mean(b_gnd))])
[Pbb,fb] = psd(b_gnd(1:Nfft),Nfft,fs,w,256);
Pbb = Pbb*norm(w)^2/sum(w)^2;
subplot(212),plot(fb/fs,10*log10(Pbb)),axis([0 0.5 -60 0])
%ylabel('Magnitude - dB')
text(5e3,-10,['Mean = ',num2str(10*log10(mean(Pbb))),' dB'])
figure(6)
subplot(211),plot(c_gnd(1:500))
%title(['Data C on channel b  (sampling freq = ',num2str(fs),' Hz)'])
text(50,15,['Max = ',num2str(max(c_gnd)),' Min = ',num2str(min(c_gnd))])
text(50,-15,['Mean = ',num2str(mean(b_gnd))])
[Pcc,fc] = psd(c_gnd(1:Nfft),Nfft,fs,w,256);
Pcc = Pcc*norm(w)^2/sum(w)^2;
subplot(212),plot(fc/fs,10*log10(Pcc))
%ylabel('Magnitude - dB')
text(5e3,-10,['Mean = ',num2str(10*log10(mean(Pcc))),' dB'])
```

E. PROGRAM LDTST3.M

```
%*****************************************************************
%
% LDTST3.M  loads data from taken from PC31 digital demodulator
%         using TST3.asm program.
%         Amux, Bmux and Cmux data are loaded and plotted.
%
%         written by; LCdr Brenner  29 Aug 94
%
%*****************************************************************
clear all, clf
Nfft = 4096;
fs = 75757.6;
w = hanning(Nfft);
%a0 = 0.35875; a1 = 0.48829; a2 = 0.14128; a3 = 0.01168; n = 0:Nfft-1;
%w = a0 - a1*cos(2*pi*n/Nfft) + a2*cos(4*pi*n/Nfft) - a3*cos(6*pi*n/Nfft);

num = input('Input data number to be loaded ','s');
filename1 = ['b:amux',num,'.dat'];
amux = ld16bit(filename1);
filename2 = ['b:bmux',num,'.dat'];
bmux = ld16bit(filename2);
filename3 = ['b:cmux',num,'.dat'];
cmux = ld16bit(filename3);
%plot the data
figure(1)
subplot(211),plot(amux(2:1000))
[Paa,fa] = psd(amux(2:Nfft),Nfft,fs,w,256);
Paa = Paa*norm(w)^2/sum(w)^2;
subplot(212),plot(fa(1:1000)/fs,10*log10(Paa(1:1000)))
SNR = 10*log10(max(Paa)/mean(Paa(100:Nfft/2)));
text(0.025,30,['SNR = ',num2str(SNR),' dB'])
figure(2)
subplot(211),plot(bmux(2:1000))
[Pbb,fb] = psd(bmux(2:Nfft),Nfft,fs,w,256);
Pbb = Pbb*norm(w)^2/sum(w)^2;
subplot(212),plot(fb(1:1000)/fs,10*log10(Pbb(1:1000))),axis([0 0.25 -50 50])
SNR = 10*log10(max(Pbb)/mean(Pbb(100:Nfft/2)));
text(0.025,30,['SNR = ',num2str(SNR),' dB'])
figure(3)
subplot(211),plot(cmux(2:1000))
[Pcc,fc] = psd(cmux(2:Nfft),Nfft,fs,w,256);
Pcc = Pcc*norm(w)^2/sum(w)^2;
subplot(212),plot(fc(1:1000)/fs,10*log10(Pcc(1:1000)))
SNR = 10*log10(max(Pcc)/mean(Pcc(100:Nfft/2)));
text(0.025,30,['SNR = ',num2str(SNR),' dB'])
```

F. PROGRAM LDTST7.M

```
%*****************************************************************
%
% LDTST7.M  loads data taken from PC31 digital demodualtor
%         using TST7.ASM program.  TST7 demodulates the signals
%         using floating point math.  Denominator, numerator and
%         output samples are stored.  4096 points of data are loaded
%         by this   program.
%
%         written by; LCdr Brenner  29 Aug 94
%
%*****************************************************************

clear all, clf, clc

num = input('Input data number to be loaded ','s');
filename1 = ['denf',num,'.dat'];
denom = ld32bit(filename1);
filename2 = ['numf',num,'.dat'];
numer = ld32bit(filename2);
filename3 = ['outf',num,'.dat'];
output = ld32bit(filename3);
N = length(output); fs = 75757.6;

%plot the data

figure(1)
L = 500; l = 0:L-1;
subplot(121),plot(l,denom(L:2*L-1))
amp = max(denom)-min(denom);
disp(['Swing= ',num2str(amp)])
subplot(122),plot(l,numer(L:2*L-1))
amp = max(numer)-min(numer);
disp(['Swing= ',num2str(amp)])

figure(2)
subplot(211),plot(output(N-L+1:N)),grid
text(50,max(output),['Amplitude= ',num2str(max(output)-min(output)),' pp'])

Nfft = 4096;
w = hanning(Nfft);
[Pxx,fx] = psd(output(N-Nfft:N),Nfft,fs,w,256);
Pxx = Pxx*norm(w)^2/sum(w)^2;
subplot(212),plot(fx(1:1000)/fs,10*log10(Pxx(1:1000))),grid
text(0.05,60,['Freq = ',num2str(find(Pxx == max(Pxx))/Nfft),' fs'])
[x,y] = ginput(6);
y = 10.^(y/20);
THD = sum(y(2:6))/y(1)*100;
```

```
text(0.05,80,['THD = ',num2str(THD),' %'])
```

G. PROGRAM TMSDEMOD.C

```
/*********************************************************************
*
*       TMSdemod.c - 4
*        This program was written for compilation using the TMS320 optimizing C compiler.
*        Data presented on MUX channels IN0 and IN1 and inputB of the DSP102 are sampled
*        and then asymmetric demoduation is performed using integer arithmatic.
*        4000 points of output data are accumulated in the vectors phase and denom.
*
*        written by LCdr Brenner 10 Mar 1994.
*         last modified : 01 May 94
*
*********************************************************************/

#include "stdio.h"                      /* constant definitions for PC31 */
#define num_samples 4000                /* number of data samples to take */
#define DECAY 0.99                      /* set integrator decay constant */

int main(void);
int denom[num_samples];                 /* denomimator term */
int output[num_samples];                /* output of algorithm */

main()
{
        volatile int* ser_gc = SER_GC;
        volatile int* adc = ADC;
        volatile int* da0 = DAC0;
        volatile int* da1 = DAC1;
        volatile int* ser_rd = SER_RD;
        volatile int* ser_td = SER_TD;
        int i, a, b, c, a_temp, b_temp, c_temp;
        int adot, bdot, cdot, numer, square, start, stop;
        a_temp = 0; b_temp =0; c_temp = 0;
        denom[0] = 0;
        output[0] = 0;

        set_ST(get_ST() | 0x0800);              /* Enable PC31 cache */
        enable_interrupts();
        timer(2,1000);                          /* set up 1Khz timebase on channel */
        enable_clock();                         /* 2 of 8254 timer */

        *ser_gc = 0x0ebd0040;                   /* initialize serial communications */
        *SER_FSX = 0x00000111;                  /* with DSP102,202, 32 bit words */
        *SER_FSR = 0x00000111;                  /* chan A MSB, chan B LSB, polled */
        *SER_CTL = 0x0000000f;                  /* operation */
        *SER_CNT = 0;
        *SER_PER = 0;

        *MUX_A = 0;                             /* Use Mux A channnel IN0, IN1 */
        *ser_td = 0;
        *da0;                                   /* set all DSP102 DACs to zero */
        while((*ser_gc & 2) == 0)               /* output voltage state */
                ;
        *ser_td = 0;
        *da1;
        while((*ser_gc & 2) == 0)
                ;
        *ser_rd;                                /* dummy reads of serial input port */
        *ser_rd;                                /* to clear for first conversion */
        *adc;                                   /* convert data on Mux channel IN0 */
        *MUX_A = 1;                             /* switch Mux to IN1 prior to next conv */
        start = uclock();

        for(i = 1; i <= num_samples; i++)       /* start of demodulation loop */
        {
                while((*ser_gc & 1) == 0)       /* wait till prev conv complete */
                        ;
                *adc;                           /* start next conv (data b,Mux 1) */
                *MUX_A = 0;                     /* switch Mux back to IN0 for data A */
                a = *ser_rd >> 16;

                while((*ser_gc & 1) == 0)       /* wait for conv( 4.8 usec) data b and c */
                        ;                       /* being sampled */
                temp = *ser_rd;                 /* read conversion data */
                b = (temp & 0xffff0000) >> 16;  /* get data b */
                c = (temp & 0xffff) >> 16;      /* get data c */

                denom[i] = (a*a + b*b + c*c) >> 16;
                adot = a - a_temp;              /* calculate derivatives */
                bdot = b - b_temp;
                cdot = c - c_temp;
                numer = a*(cdot - bdot) + b*(adot - cdot) + c*(bdot - adot);

                output[i] = DECAY*output[i-1] + numer/denom[i];
                a_temp = a; b_temp = b; c_temp = c;
        }
        stop = uclock();
        printf("\nstart time = %i", start);
        printf("\nstop time = %i", stop);
        printf("\ntime taken = %i", (stop - start) );
}
```

H. PROGRAM TST3.ASM

```
        .title "Data sampling using Mux"
*****************************************************
*
* TST3.ASM - Routine to take data from DSP102 at Max
*       sampling rate using Multiplexed input on input A of DSP102.
*       Data is stored in vectors a_dat, b_dat and c_dat.
*       fs = 75,757.6 Hz
*
*****************************************************
*
* Reset and interrupt vector table specification.  This
* arrangement assumes that during linking, the following
* text segment will be placed to start at the origin of the
* vector table.
*

        .global reset,start
        .global rint0,int06,XRPT

        .sect   "MC_vec"        ; Named section
reset   .word   start           ; Hardware reset vector
*
        .word   XRPT            ; EI0                   (01)
        .word   XRPT            ; EI1                   (02)
        .word   XRPT            ; EI2                   (03)
        .word   XRPT            ; EI3                   (04)

        .word   XRPT            ; Serial port 0 XMT     (05)
rint0   .word   int06           ; Serial port 0 RCV     (06)

        .word   XRPT            ; Serial port 1 XMT     (07)
        .word   XRPT            ; Serial port 1 RCV     (08)

        .word   XRPT            ; Timer 0               (09)
        .word   XRPT            ; Timer 1               (10)

        .word   XRPT            ; DMA                   (11)

        .space  20               ; Reserved

        .space  32               ; Space for the 32 traps (32-63)

*
* Entry point for TST3 routine
* Initialization routine
*
;
; constants for intialization routines
```

```
        .data
base    .word   00808000h          ; base address of onchip peripherals
ser_ini .word   0ebc0040h          ; serial data bus initialization word
ser_gc  .word   00808040h          ; base address of serial port 0
ser_rd  .word   0080804ch          ; address of serial port receive data
stack   .word   00809810h          ; address of start of stack
blk0    .word   00809800h          ; start address of internal RAM block 0
blk1    .word   00809c00h          ; start address of internal RAM block 1

        .text
; Reconfigure Primary Bus Control Register for PC31

start   LDI     @base,AR0          ; Load AR0 with base addr
        LDI     1090h,R1           ; Init PBCR for SWW=2,
        STI     R1,*+AR0(64h)      ; and WTCNT = 4.
;
; intilaize serial port 0
        LDI     @ser_ini,R0        ; setup for serial port transfers
        STI     R0,*+AR0(40h)      ; with DSP102 at 8.33333 Mhz
        LDI     111h,R0            ; 32 bit words the first 16 MSB bits
        STI     R0,*+AR0(42h)      ; are data A and the 16 LSB bits are B
        STI     R0,*+AR0(43h)
        LDI     0fh,R0
        STI     R0,*+AR0(44h)
        LDI     0,R0
        STI     R0,*+AR0(45h)
        STI     R0,*+AR0(46h)
;
; zero internal memory
        LDI     @blk0,AR0          ; AR0 points to RAM block 0
        LDI     @blk1,AR1          ; AR1 points to RAM block 1
        LDI     0,R0               ; load intialization value 00
        RPTS    1023               ; repeat 1023 times
        STI     R0,*AR0++(1)       ; zero RAM block 0
||      STI     R0,*AR1++(1)       ; zero RAM block 1
;
;  intialize  and start internal timer 1
        LDI     @base,AR0          ; base address of on-chip peripherals
        STI     R0,*+AR0(30h)      ; stop timer 1
        LDI     55,R0              ; setup count for 151,515.2 Hz
        STI     R0,*+AR0(38h)
        LDI     2c1h,R0            ; setup for 1 cycle pluse
        STI     R0,*+AR0(30h)      ; and start timer 1
;
; intialize stack pointer
        LDI     @stack,SP          ; intialize stack pointer
        LDI     @MUX,AR0           ; load address of Multiplexer
        LDI     @Mux,AR2           ; load address of mux shadow register
        LDI     0,R0
        STI     R0,*AR0            ; set mux channel IN0
||      STI     R0,*AR2            ; update mux shadow reg
```

```
;
; clear first two samples of ADC before starting loop
        LDI     @ser_rd,AR1             ; load address of serial port0 receive
wait1   TSTB    20h,IF                  ; wait for serial port interrupt
        BZ      wait1
        LDI     1,R2                    ;
        STI     R2,*AR0                 ; change mux to channel IN1
        LDI     *AR1,R1                 ; dummy read of serial data
        STI     R1,@dummy1              ; store at dummy1
        XOR     IF,IF                   ; clear interrupt flag
wait2   TSTB    20h,IF                  ; wait for next interrupt
        BZ      wait2
        STI     R0,*AR0                 ; change mux back to channel IN0
        LDI     *AR0,R1                 ; dummy read
        STI     R1,@dummy2              ; store at dummy2
        XOR     IF,IF                   ; clear interrupt flag
;
; load registers for the interrupt service routine
        LDP     0                       ; load data page to point to bss section
        LDI     @A_dat,AR3              ; AR3 points to data A
        LDI     @B_dat,AR4              ; AR4 points to data B
        LDI     @C_dat,AR5              ; AR5 points to data C
        LDI     @MASK,RC                ; RC has mask value
        LDI     0,IR0                   ; zero data index pointer

        LDI     20h,IE                  ; enable serial port0 receive interrupt
        LDI     2800h,ST                ; and enable global interrupts and cache
;
; go into infinite loop
loop:
        B       loop
;
; Interrupt service routine to read ADC conversion data
int06:  IACK    *AR2                    ; interrupt acknowledge signal
        LDI     *AR1,R1                 ; read data from serial port
        LDI     *AR2,R0                 ; check if mux is set to channel IN0
        BNZ     int06a
        ASH     -16,R1                  ; store upper 16 MSBs as data A
        BD       int06b                 ; return when read
        STI     R1,*+AR3(IR0)           ; and store it away
        LDI     1,R0                    ;
        STI     R0,*AR0                 ; change mux channel to IN0
||      STI     R0,*AR2                 ; update shadow register
int06a  LDI     0,R2
        STI     R2,*AR0                 ; change mux to channel IN1
||      STI     R2,*AR2                 ; update mux shadow register
        LDI     R1,R2
        ASH     -16,R1                  ; shift 16 MSBs to the right
        STI     R1,*+AR4(IR0)           ; store B data
        AND     RC,R2                   ; mask off lower 16 LSB
        ASH     16,R2                   ; rotate 16 bits left to set sign
```

```
         ASH    -16,R2                    ; rotate back preserving sign
         STI    R2,*+AR5(IR0)             ; store C data
         ADDI   1,IR0                     ; increment data pointer
         CMPI   @SIZE,IR0                 ; at end of data buffer yet?
         LDIZ   0,IR0                     ; if so set index pointer to start
int06b:  IACK   *AR2                      ; interrupt acknowledge
         RETI
;
; Interrupt service routine for unspecified interrupts
XRPT:
         RETI
;
; program data definition
         .globl size
         .globl a,b,c
         .bss   a,2000h
         .bss   b,2000h
         .bss   c,2000h
         .globl mux
         .bss   mux,1
         .globl dummy1
         .bss   dummy1,1
         .globl dummy2
         .bss   dummy2,1

         .data
MASK     .word  0ffffh
SIZE     .word  2000h
A_dat    .word  a
B_dat    .word  b
C_dat    .word  c
Mux      .word  mux
MUX      .word  0fffa80h

         .end
```

I. PROGRAM TST5.ASM

```
        .title    "Multiplexed demodulator using integer math"
***********************************************************
*
* TST5.ASM - Digital demodulation algortihm using integer operations.
*       Input data is sampled at Max rate using Multiplexed inputs
*       on input A of the DSP102.  8192 points of Denominator
*       numerator and output data are stored.
*
***********************************************************
* Reset and interrupt vector table specification.  This
* arrangement assumes that during linking, the following
* text segment will be placed to start at the origin of the
* vector table.
*

        .global   reset,start
        .global   rint0,int06,XRPT
        .ref      DIV_I,DIV_U

        .sect     "MC_vec"          ; Named section
reset   .word     start             ; Hardware reset vector
*
        .word     XRPT              ; EI0                (01)
        .word     XRPT              ; EI1                (02)
        .word     XRPT              ; EI2                (03)
        .word     XRPT              ; EI3                (04)

        .word XRPT                  ; Serial port 0 XMT  (05)
rint0   .word int06                 ; Serial port 0 RCV  (06)

        .word     XRPT              ; Serial port 1 XMT  (07)
        .word     XRPT              ; Serial port 1 RCV  (08)

        .word     XRPT              ; Timer 0            (09)
        .word     XRPT              ; Timer 1            (10)

        .word     XRPT              ; DMA                (11)

        .space    20                ; Reserved

        .space    32                ; Space for the 32 traps (32-63)

; Entry point for TST5 routine
; Initialization
;
; constants for intialization routines
        .data
base    .word     00808000h
```

```
ser_ini  .word   0ebc0040h                   ; serial data bus initialization word
ser_gc   .word   00808040h                   ; base address of serial port 0
ser_rd   .word   0080804ch                   ; address of received data
stack    .word   00809810h                   ; address of start of stack
blk0     .word   00809800h                   ; start address of internal RAM block 0
blk1     .word   00809c00h                   ; start address of internal RAM block 1

         .text
; Reconfigure Primary Bus Control Register for PC31
;
start    LDI     @base,AR0                   ; Load AR0 with base addr
         LDI     1090h,R1                    ; Init PBCR for SWW=2,
         STI     R1,*+AR0(64h)               ; and WTCNT = 4.
;
; intilaize serial port 0
         LDI     @ser_ini,R0                 ; setup for serial port transfers
         STI     R0,*+AR0(40h)               ; with DSP102 at 8.33333 Mhz
         LDI     111h,R0                     ; 32 bit words the first 16 MSB bits
         STI     R0,*+AR0(42h)               ; are data A and the 16 LSB bits are B
         STI     R0,*+AR0(43h)
         LDI     0fh,R0
         STI     R0,*+AR0(44h)
         LDI     0,R0
         STI     R0,*+AR0(45h)
         STI     R0,*+AR0(46h)
;
; zero internal memory
         LDI     @blk0,AR0                    ; AR0 points to RAM block 0
         LDI     @blk1,AR1                    ; AR1 points to RAM block 1
         LDI     0,R0                        ; load intialization value 00
         RPTS    1023                        ; repeat 1023 times
         STI     R0,*AR0++(1)                ; zero RAM block 0
||       STI     R0,*AR1++(1)                ; zero RAM block 1
;
; intialize internal timer 1
         LDI     @base,AR0                    ; base address of peripherals
         STI     R0,*+AR0(30h)               ; stop timer 1
         LDI     60,R0                       ; setup count for 151.5 Khz
         STI     R0,*+AR0(38h)
         LDI     2c1h,R0                     ; setup for 1 cycle pluse
         STI     R0,*+AR0(30h)               ; and start timer 1
;
; intialize stack pointer and multiplexer
         LDI     @stack,SP                    ; intialize stack pointer
         LDI     @MUX,AR0                    ;
         LDI     0,R0                        ;
         STI     R0,*AR0                     ; set multiplexer to channel IN0
         STI     R0,@mux                     ; update mux shadow register
;
; Clear first two samples from ADC and get initial samples
; for A,B and C to calculate initial first derivatives
```

```
;
        LDI     @ser_gc,AR1             ; base address of serial port 0
        LDI     1,R0
wait1   TSTB    R0,*AR1                 ; wait for a data conversion
        BZ      wait1                   ; perform 2 dummy reads and then get
        LDI     *+AR1(0ch),R1           ; initialization data
wait2   TSTB    R0,*AR1                 ; wait for next conversion
        BZ      wait2
        LDI     *+AR1(0ch),R1           ; dummy read of data
wait3   TSTB    R0,*AR1                 ; wait for another conversion
        BZ      wait3                   ;
        LDI     *+AR1(0ch),R1           ; read data from serial port 0
        STI     R0,*AR0                 ; change multiplexer to channel IN1
        ASH     -16,R1                  ; get upper 16 MSBs
        STI     R1,@aprev               ; store as previous sample for derivative
wait4   TSTB    R0,*AR1                 ; wait for next conversion
        BZ      wait4                   ;
        LDI     *+AR1(0ch),R1           ;
        LDI     0,R3                    ; switch multiplexer back to IN0
        STI     R3,*AR0                 ;
        LDI     R1,R2                   ; save data temporarily
        ASH     -16,R1                  ; get upper 16 MSBs
        LDI     R1,R4
        STI     R1,@bprev               ; store B sample for derivative calculation
        AND     0ffffh,R2               ; mask off lower 16 LSBs
        ASH     16,R2                   ;
        ASH     -16,R2                  ; preserve sign of result
        STI     R2,@cprev               ; store C sample for previous value
wait5   TSTB    R0,*AR1                 ; wait for next conversion
        BZ      wait5                   ;
        LDI     *+AR1(0ch),R1           ; read data
        STI     R0,*AR0                 ; change mux to channel IN1
        ASH     -16,R1                  ; store A data in R1
        STI     R1,@asave               ; store A at asave
wait6   TSTB    R0,*AR1                 ; wait for next conversion
        BZ      wait6                   ;
        LDI     *+AR1(0ch),R0           ;
        LDI     0,R2
        STI     R2,*AR0                 ; switch mux back to IN0
        LDI     R0,R2                   ;
        ASH     -16,R2                  ; data B in R2
        STI     R2,@bsave               ; store B data at bsave
        AND     0ffffh,R0               ; mask off C data
        ASH     16,R0                   ; shift left 16 bits to set sign
        ASH     -16,R0                  ; renormalize data, data C in R0
        STI     R0,@csave               ; save data C at csave
        LDI     @Prev,AR7               ; base address of previous data samples
        SUBI    *AR7,R1,R3              ; adot = (asave - aprev) adot in R3
        STI     R3,@adot                ; store adot
        STI     R1,*AR7++               ; update aprev
        SUBI    *AR7,R2,R4              ; bdot = (bsave - bprev) bdot in R4
        STI     R4,@bdot                ; store bdot
        STI     R2,*AR7++               ; update bprev
        SUBI    *AR7,R0,R5              ; cdot = (csave - cprev) cdot in R5
        STI     R5,@cdot                ; store cdot
        STI     R0,*AR7                 ; update cprev
        MPYI    R1,R1                   ; square A data
        MPYI    R2,R2                   ; square B data
        MPYI    R0,R0                   ; square C data
        ADDI    R0,R2                   ; C^2 + B^2
        ADDI    R2,R1                   ; C^2 + B^2 + A^2  denom in R1
        ASH     -16,R1                  ; scale denominator by 65536
        LDI     @Denom,AR2
        STI     R1,*AR2                 ; store denominator
        SUBI    R3,R4,R2                ; (bdot - abot) save in R2
        MPYI    R2,*AR7--,R2            ; numer3 = c*(bdot - adot) in R2
        ASH     -1,R2                   ; divide numer3 by 2
        SUBI    R5,R3,R6                ; (adot - cdot) save in R6
        MPYI    R6,*AR7--,R6            ; numer2 = b*(adot - cdot) in R6
        ASH     -1,R6                   ; divide numer2 by 2
        SUBI    R4,R5,R0                ; (cdot - bdot) save in R0
        MPYI    R0,*AR7,R0              ; numer1 = a*(cdot - bdot) in R0
        ASH     -1,R0                   ; divide numer1 by 2
        ADDI    R2,R6                   ; numer2 + numer3
        ADDI    R6,R0                   ; numer=numer1+numer2+numer3 in R0
        LDI     @Numer,AR2              ; base address of numerator data
        STI     R0,*AR2                 ; store numerator data
        CALL    DIV_I                   ; Divide R0/R1
        LDI     @Output,AR2             ; load base address of output
        STI     R0,@last                ; initialize last output value
        STI     R0,*AR2                 ; store at first output value
; load registers for the interrupt service routine
;
        LDP     0                       ; load data page to point to bss section
        LDI     @MUX,AR0                ; AR0 has address of multiplexer
        LDI     @ser_rd,AR1             ; AR1 has address of serial read data
        LDI     @Mux,AR2                ; AR2 has address of mux shadow reg
        LDI     @Denom,AR3              ; AR3 points to denominator data
        LDI     @Numer,AR4              ; AR4 points to numerator data
        LDI     @Output,AR5             ; AR5 points to output data
        LDI     @MASK,BK                ; BK has mask value
        LDI     1,IR0                   ; zero data index pointer

        LDI     20h,IE                  ; enable serial port0 receive interrupt
        LDI     2800h,ST                ; and enable global interrupts and cache
; go into infinite loop
;
loop:
        B       loop
;
; serial port 0 interrupt serive routine
int06:  LDI     2800h,ST                ; reenable interrupts
```

```
        LDI     *AR1,RS              ; read data from serial port 0
        LDI     *AR2,R7              ; check mux shadow reg
        BNZ     int06a               ;
        ASH     -16,RS               ; mask off LSB 16 bits
        BD      int06b               ;
        STI     RS,@asave            ; store A data at asave
        LDI     1,R7                 ;
        STI     R7,*AR0              ; change mux to IN0
||      STI     R7,*AR2              ; update mux shadow reg
int06a  LDI     0,R1                 ;
        STI     R1,*AR0              ; change mux to IN1
||      STI     R1,*AR2              ; update mux shadow register
        LDI     RS,R1                ;
        ASH     -16,R1               ;
        LDI     R1,R4                ; save data B in R4
        MPYI    R1,R1                ; save B^2 in R1
        AND     BK,RS,R2             ; mask off C data
        ASH     16,R2                ; shift left 16 bits to set sign
        ASH     -16,R2               ; renormalize data
        LDI     R2,R5                ; save data C in R5
        MPYI    R2,R2                ; save c^2 in R2
        LDI     @asave,R3            ; restore data A
        LDI     R3,R6                ; save data A in R6
        MPYI    R3,R3                ; save a^2 in R3
        ADDI    R3,R1                ; (a^2 + b^2)
        ADDI    R2,R1                ; (a^2 + b^2 + c^2)
        ASH     -16,R1               ; divide denominator by 2^16
        STI     R1,*+AR3(IR0)        ; store and keep denominator in R1
        LDI     @Prev,AR7            ; base address of previous data samples
        SUBI    *AR7,R6,R0           ; adot = (asave - aprev) adot in R0
        STI     R6,*AR7++            ; update aprev
        SUBI    *AR7,R4,R2           ; bdot = (bsave - bprev) bdot in R2
        STI     R4,*AR7++            ; update bprev
        SUBI    *AR7,R5,R3           ; cdot = (csave - cprev) cdot in R3
        STI     R5,*AR7              ; update cprev
        SUBI    R0,R2,R4             ; (bdot - abot) save in R4
        MPYI    R4,*AR7--,R4         ; numer3 = c*(bdot - adot) in R4
        ASH     -1,R4                ; divide numer1 by 2
        SUBI    R3,R0,R5             ; (adot - cdot) save in R5
        MPYI    R5,*AR7--,R5         ; numer2 = b*(adot - cdot) in R5
        ASH     -1,R5                ; divide numer2 by 2
        SUBI    R2,R3,R6             ; (cdot - bdot) save in R6
        MPYI    R6,*AR7,R6           ; numer1 = a*(cdot - bdot) in R6
        ASH     -1,R6                ; divide numer3 by 2
        ADDI    R5,R6                ; numer2 + numer3
        ADDI    R6,R4,R0             ; numer=numer1+numer2+numer3 in R0
        LDI     @Numer,AR4           ; base address of numerator data
        STI     R0,*+AR4(IR0)        ; store numerator data
        CALL    DIV_I                ; Divide R0/R1
        FLOAT   R0                   ; convert to floating point
        LDI     @last,R1             ; load previous output
```

```
         FLOAT R1                       ; convert to floating point
         MPYF  @DECAY,R1                ; Multiply decay*last output
         ADDF  R1,R0
         FIX   R0,R2
         LDI   @Output,AR5              ; get pointer to output data
         STI   R2,*+AR5(IR0)            ; store output
         STI   R2,@last                 ; update last output value
         ADDI  1,IR0                    ; increment data pointer
         CMPI  @SIZE,IR0                ; at end of data buffer yet?
         LDIZ  0,IE                     ; if so disable interrupts
int06b:
         RETI

; Interrupt service routine for unspecified interrupts
;
XRPT:
         RETI
;
; system data definitions
         .globl   denom
         .bss     denom,2000h
         .globl   numer
         .bss     numer,2000h
         .globl   output
         .bss     output,2000h
         .globl   asave,bsave,csave
         .bss     asave,1
         .bss     bsave,1
         .bss     csave,1
         .globl   aprev
         .bss     aprev,1
         .globl   bprev
         .bss     bprev,1
         .globl   cprev
         .bss     cprev,1
         .globl   last
         .bss     last,1
         .globl   mux
         .bss     mux,1
         .globl   adot,bdot,cdot
         .bss     adot,1
         .bss     bdot,1
         .bss     cdot,1

         .data
MASK     .word    0ffffh
DECAY    .float   9.9e-1
SIZE     .word    2000h
Denom    .word    denom
Numer    .word    numer
Output   .word    output
```

```
MUX     .word   0fffa80h
Mux     .word   mux
Prev    .word   aprev
Save    .word   asave
        .end
```

J. PROGRAM TST7.ASM

```
        .title    "Multiplexed demodulator using floating point"
*****************************************************************
*
* TST7.ASM - Demodulation routine taking data from DSP102
*         at Max sampling rate using Multiplexed inputs. Demodulation
*         is done using floating point operations.  8192 (decimal) points of the
*         denominator, numerator and output signals are computed and stored.
*
*****************************************************************
* Reset and interrupt vector table specification.  This
* arrangement assumes that during linking, the following
* text segment will be placed to start at the origin of the
* vector table.
*

        .global  reset,start
        .global  rint0,int06,XRPT
        .ref     DIV_F

        .sect    "MC_vec"         ; Named section
reset   .word    start            ; Hardware reset vector
*
        .word    XRPT             ; EI0                    (01)
        .word    XRPT             ; EI1                    (02)
        .word    XRPT             ; EI2                    (03)
        .word    XRPT             ; EI3                    (04)

        word     XRPT             ; Serial port 0 XMT      (05)
rint0   word     int06            ; Serial port 0 RCV      (06)

        .word    XRPT             ; Serial port 1 XMT      (07)
        .word    XRPT             ; Serial port 1 RCV      (08)

        .word    XRPT             ; Timer 0                (09)
        .word    XRPT             ; Timer 1                (10)

        .word    XRPT             ; DMA                    (11)

        .space   20               ; Reserved

        .space   32               ; Space for the 32 traps (32-63)

; Entry point for TST7 routine
; Initialization
;
; constants for intialization routines
        .data
base    .word    00808000h               ; base address of onboard peripherals
```

```
ser_ini  .word   0ebc0040h               ; serial data bus initialization word
ser_gc   .word   00808040h               ; base address of serial port 0
ser_rd   .word   0080804ch               ; address of received data
stack    .word   00809810h               ; address of start of stack
blk0     .word   00809800h               ; start address of internal RAM block 0
blk1     .word   00809c00h               ; start address of internal RAM block 1

         .text
; Reconfigure Primary Bus Control Register for PC31
;
start    LDI     @base,AR0               ; Load AR0 with base addr
         LDI     1090h, R1               ; Init PBCR for SWW=2,
         STI     R1,*+AR0(64h)           ; and WTCNT = 4.
;
; intilaize serial port 0
;
         LDI     @ser_ini,R0             ; setup for serial port transfers
         STI     R0,*+AR0(40h)           ; with DSP102 at 8.33333 Mhz
         LDI     111h,R0                 ; 32 bit words the first 16 MSB bits
         STI     R0,*+AR0(42h)           ; are data A and the 16 LSB bits are B
         STI     R0,*+AR0(43h)
         LDI     0fh,R0
         STI     R0,*+AR0(44h)
         LDI     0,R0
         STI     R0,*+AR0(45h)
         STI     R0,*+AR0(46h)
;
; zero internal memory
         LDI     @blk0,AR0               ; AR0 points to RAM block 0
         LDI     @blk1,AR1               ; AR1 points to RAM block 1
         LDI     0,R0                    ; load intialization value 00
         RPTS    1023                    ; repeat 1023 times
         STI     R0,*AR0++(1)            ; zero RAM block 0
||       STI     R0,*AR1++(1)            ; zero RAM block 1
;
; intialize internal timer 1
         LDI     @base,AR0               ; base address of peripherals
         STI     R0,*+AR0(30h)           ; stop timer 1
         LDI     55,R0                   ; setup count for 151,515.2 Hz
         STI     R0,*+AR0(38h)
         LDI     2c1h,R0                 ; setup for 1 cycle pluse
         STI     R0,*+AR0(30h)           ; and start timer 1
;
; intialize stack pointer and multiplexer
         LDI     @stack,SP               ; intialize stack pointer
         LDI     @MUX,AR2                ;
         LDI     0,R0                    ;
         STI     R0,*AR2                 ; set multiplexer to channel IN0
         STI     R0,@mux                 ; update mux shadow register
;
; Clear first two samples from ADC and get values to
; initialize first couputations
;
          LDI     @ser_gc,AR1            ; base address of serial port 0
          LDI     1,R0
wait1     TSTB    R0,*AR1                ; wait for a data conversion
          BZ      wait1                  ; perform 2 dummy reads and then get
          LDI     *+AR1(0ch),R1          ; initialization data
wait2     TSTB    R0,*AR1                ; wait for next conversion
          BZ      wait2
          LDI     *+AR1(0ch),R1          ; dummy read of data
wait3     TSTB    R0,*AR1                ; wait for another conversion
          BZ      wait3                  ;
          LDI     *+AR1(0ch),R1          ; read data from serial port 0
          STI     R0,*AR2                ; change multiplexer to channel IN1
          ASH     -16,R1                 ; get upper 16 MSBs
          FLOAT   R1
          STF     R1,@aprev              ; store as previous sample for derivative
wait4     TSTB    R0,*AR1                ; wait for next conversion
          BZ      wait4                  ;
          LDI     *+AR1(0ch),R1          ;
          LDI     0,R3                   ; switch multiplexer back to IN0
          STI     R3,*AR2                ;
          LDI     R1,R2                  ; save data temporarily
          ASH     -16,R1                 ; get upper 16 MSBs
          FLOAT   R1                     ; convert to floating pt
          STF     R1,@bprev              ; store B sample for derivative calculation
          AND     0ffffh,R2              ; mask off lower 16 LSBs
          ASH     16,R2                  ;
          ASH     -16,R2                 ; preserve sign of result
          FLOAT   R2                     ; convert ot floating pt
          STF     R2,@cprev              ; store C sample for previous value
wait5     TSTB    R0,*AR1                ; wait for next conversion
          BZ      wait5                  ;
          LDI     *+AR1(0ch),R1          ; read data
          STI     R0,*AR2                ; change mux to channel IN1
          ASH     -16,R1                 ; store A data in R1
          FLOAT   R1                     ; convert data to floating pt
wait6     TSTB    R0,*AR1                ; wait for next conversion
          BZ      wait6                  ;
          LDI     *+AR1(0ch),R0          ;
          LDI     0,R2                   ;
          STI     R2,*AR2                ; switch mux back to IN0
          LDI     R0,R2                  ;
          ASH     -16,R2                 ; data B in R2
          FLOAT   R2                     ; convert data B to floating pt
          AND     0ffffh,R0              ; mask off C data
          ASH     16,R0                  ; shift left 16 bits to set sign
          ASH     -16,R0                 ; renormalize data, data C in R0
          FLOAT   R0                     ; convert data C to floating pt
          LDI     @Prev,AR7              ; base address of previous data samples
          SUBF    *AR7,R1,R3             ; adot = (asave - aprev) adot in R3
```

```
        STF     R1,*AR7++               ; update aprev
        SUBF    *AR7,R2,R4              ; bdot = (bsave - bprev) bdot in R4
        STF     R2,*AR7++               ; update bprev
        SUBF    *AR7,R0,R5              ; cdot = (csave - cprev) cdot in R5
        STF     R0,*AR7                 ; update cprev
        MPYF    R1,R1                   ; square A data
        MPYF    R2,R2                   ; square B data
        MPYF    R0,R0                   ; square C data
        ADDF    R0,R2                   ; C^2 + B^2
        ADDF    R2,R1                   ; C^2 + B^2 + A^2  denom in R1
        SUBF    R3,R4,R2                ; (bdot - abot) save in R2
        MPYF    R2,*AR7--,R2            ; numer3 = c*(bdot - adot) in R2
        SUBF    R5,R3,R6                ; (adot - cdot) save in R6
        MPYF    R6,*AR7--,R6            ; numer2 = b*(adot - cdot) in R6
        SUBF    R4,R5,R0                ; (cdot - bdot) save in R0
        MPYF    R0,*AR7,R0              ; numer1 = a*(cdot - bdot) in R0
        ADDF    R2,R6                   ; numer2 + numer3
        ADDF    R6,R0                   ; numer=numer1+numer2+numer3 in R0
        CALL    DIV_F                   ; Divide R0/R1
        STF     R0,@last                ; initialize last output value
        LDI     @Output,AR2             ; load base address of output
        MPYF    @SCALE,R0               ; scale output before conversion to int
        FIX     R0                      ; convert output to integer
        STI     R0,*AR2                 ; store at first output value
;
; load registers for the interrupt service routine
;
        LDP     0                       ; load data page to point to bss section
        LDI     @MUX,AR0                ; AR0 has address of multiplexer
        LDI     @ser_rd,AR1             ; AR1 has address of serial read data
        LDI     @Mux,AR2                ; AR2 has address of mux shadow reg
        LDI     @Output,AR3             ; AR5 points to output data
        LDI     @MASK,BK                ; BK has mask value
        LDI     1,IR0                   ; zero data index pointer

        LDI     20h,IE                  ; enable serial port0 receive interrupt
        LDI     2800h,ST                ; and enable global interrupts and cache
;
; go into infinite loop
;
loop:
        B       loop
;
; serial port 0 interrupt serive routine
int06:  LDI     2800h,ST                ; reenable interrupts
        LDI     *AR1,RS                 ; read data from serial port 0
        LDI     *AR2,R7                 ; check mux shadow reg
        BNZ     int06a                  ;
        ASH     -16,RS                  ; mask off LSB 16 bits
        FLOAT RS,R7
        BD       int06b                 ;
        STF     R7,@asave           ; store A data at asave
        LDI     1,R7                ;
        STI     R7,*AR0             ; change mux to IN0
||      STI     R7,*AR2             ; update mux shadow reg
int06a  LDI     0,R1                ;
        STI     R1,*AR0             ; change mux to IN1
||      STI     R1,*AR2             ; update mux shadow register
        LDI     RS,R1               ;
        ASH     -16,R1              ;
        FLOAT R1
        LDF     R1,R4               ; save data B in R4
        MPYF    R1,R1               ; save B^2 in R1
        AND     BK,RS,R2            ; mask off C data
        ASH     16,R2               ; shift left 16 bits to set sign
        ASH     -16,R2              ; renormalize data
        FLOAT R2
        LDF     R2,R5               ; save data C in R5
        MPYF    R2,R2               ; save c^2 in R2
        LDF     @asave,R3           ; restore data A
        LDF     R3,R6               ; save data A in R6
        MPYF    R3,R3               ; save a^2 in R3
        ADDF    R3,R1               ; (a^2 + b^2)
        ADDF    R2,R1               ; (a^2 + b^2 + c^2)
        LDI     @Prev,AR7           ; base address of previous data samples
        SUBF    *AR7,R6,R0          ; adot = (asave - aprev) adot in R0
        STF     R6,*AR7++           ; update aprev
        SUBF    *AR7,R4,R2          ; bdot = (bsave - bprev) bdot in R2
        STF     R4,*AR7++           ; update bprev
        SUBF    *AR7,R5,R3          ; cdot = (csave - cprev) cdot in R3
        STF     R5,*AR7             ; update cprev
        SUBF    R0,R2,R4            ; (bdot - abot) save in R4
        MPYF    R4,*AR7--,R4        ; numer3 = c*(bdot - adot) in R4
        SUBF    R3,R0,R5            ; (adot - cdot) save in R5
        MPYF    R5,*AR7--,R5        ; numer2 = b*(adot - cdot) in R5
        SUBF    R2,R3,R6            ; (cdot - bdot) save in R6
        MPYF    R6,*AR7,R6          ; numer1 = a*(cdot - bdot) in R6
        ADDF    R5,R6               ; numer2 + numer3
        ADDF    R6,R4,R0            ; numer=numer1+numer2+numer3 in R0
        CALL    DIV_F               ; Divide R0/R1
        LDF     @last,R1            ; load previous output
        MPYF    @DECAY,R1           ; Multiply decay*last output
        ADDF    R1,R0               ;
        STF     R0,@last            ; update last output sample
        MPYF    @SCALE,R0           ; scale output for integer value
        FIX     R0                  ; convert to integer
        STI     R0,*+AR3(IR0)       ; store integer output
        ADDI    1,IR0               ; increment data pointer
        CMPI    @SIZE,IR0           ; at end of data buffer yet?
        LDIZ    0,IE                ; if so disable interrupts
int06b:
        RETI
```

```
; Interrupt service routine for unspecified interrupts
;
XRPT:
          RETI
;
; system data definitions
          .globl    denom
          .bss      denom,2000h
          .globl    numer
          .bss      numer,2000h
          .globl    output
          .bss      output,2000h
          .globl    asave
          .bss      asave,1
          .globl    aprev
          .bss      aprev,1
          .globl    bprev
          .bss      bprev,1
          .globl    cprev
          .bss      cprev,1
          .globl    last
          .bss      last,1
          .globl    mux
          .bss      mux,1


          .data
MASK      .word     0ffffh
DECAY     .float    9.9e-1
SIZE      .word     2000h
SCALE     .float    30000.0
Denom     .word     denom
Numer     .word     numer
Output    word      output
MUX       .word     0fffa80h
Mux       .word     mux
Prev      .word     prev
Save      .word     save

          .end
```

REFERENCES.

1. D. A. Brown, T. Hofler, and S. L. Garrett, "High-sensitivity, fiber-optic, flexural disk hydrophone with reduced acceleration response," in *Fiber and Integrated Optics*, Vol. 8, pp 169-191, 1989.

2. T. Hofler and S. L. Garrett, Flexural disk fiber-optic interferometric omnidirectional hydrophone, U. S. Patent No. 4,959,539, September 25, 1990.

3. S. L. Garrett, D. A. Brown, B. L. Beaton, K.Wetterskog, and J. Serocki, "A general purpose fiber-optic hydrophone made of castable epoxy," in *Fiber Optic and Laser Sensor VIII*, (SPIE), Vol. 1367, pp 13-29, 1990.

4. S. L. Garrett and D. A. Danielson, Fiber-optic interferometric ellipsodial flextensional hydrophone, U. S. Patent No. 4,951,271, August 21, 1990.

5. D. A. Danielson and S. L. Garrett, "Fiber-optic ellipsodial flextensional hydrophones,", in *Journal of Lightwave and Tech.*, Vol 7(12), pp 1995-2002, 1989.

6. E. Udd, "Fiber Optic Sensors based on the Mach-Zehnder and Michelson Interferometers," in *Fiber Optic Sensors: An Introduction for Scientists and Engineers,* Wiley, New York, 1991.

7. C. B. Cameron, "Recovering Signals from Optical Fiber Interferometric Sensors," Ph. D Dissertation, Naval Postgraduate School, Monterey CA, June 1991.

8. D. A Brown, C. B. Cameron, R. M. Keolian, D. L. Gardner, and S. L. Garrett, "A Symmetric 3x3 Coupler Based Demodulator for Fiber Optic Interferometric Sensors," in *Proceedings of Fiber Optic and Laser Sensors IX*, (SPIE), Vol. 1585 pp 328-335, 1991.

9. B. R. McGinnis, "Digital Demodulation of an Interferometric Signal," Masters Thesis, Naval Postgraduate School, Monterey, CA, June 1993.

10. Texas Instruments, *TMS320C3x User's Guide*, Revision E, June 1991.

11. Texas Instruments, *TMS320 Floating-Point DSP Optimizing C Compiler, User's Guide*, Revision A, October 1991.

12. Texas Instruments, *TMS320 Floating-Point DSP Assembly Language tools, User's Guide*, Revision A, September 1991.

13. Burr-Brown, *Integrated Circuits Data Book Supplement, Volume 33c*, Burr-Brown Corporation, 1992.

14. Turbo C, version 2.0, Borland International, Scotts Valley, CA.

15. MATLAB for Windows version 4.0, The MathWorks, Inc., Natick, MA.

16. Innovation Integration, *PC31 C System, User's Guide*, Innovation Integration, Moorepark, CA, 1993.

17. J. Harris, "Harris: Use of Windows for Harmonic Analysis.", in, *Proc. of the IEEE.*, Vol 66, No 1, January 1978.

18. Burr-Brown, *Integrated Circuits Data Book, Volume 33*, Burr-Brown Corporation, 1989.

19. G. R. Reid, "Multiplexed Fiber-Optic Sensors," Masters Thesis, Naval Postgraduate School, Monterey, CA, December 1993.

20. B.W. Kernigham and D. M. Ritchie, *The C Programming Language,* Prentice Hall, New Jersey, 1988.

21. H. W. Ott, *Noise Reduction Techniques in Electronic Systems*, John Wiley and Sons, New York, 1988, Chapters 8 and 10.

22. Analog Devices, *High Speed Design Seminar*, 1990.

23. P. Horowitz and W. Hill, *The Art of Electronics,* Cambridge University Press, New York, 1984.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
Cameron Station
Alexandria, VA, 22304-6145

2. Library, Code 52 2
Naval Postgraduate School
Monterey, CA, 93943-5102

3. Prof. R. M. Keolian, Code PH/Kn 2
Physics Department
Naval Postgraduate School
Monterey, CA, 93943-5000

4 Prof. S. L. Garrett, Code PH/Gx 1
Physics Department
Naval Postgraduate School
Monterey, CA, 93943-5000

5. Brown University 1
Professor David A. Brown
Division of Engineering
Box D
182 Hope St.
Providence, RI 02912

6. LCdr David W. Brenner 1
620 Du Buisson Cres.
Orleans, Ontario, Canada K4A 3A3